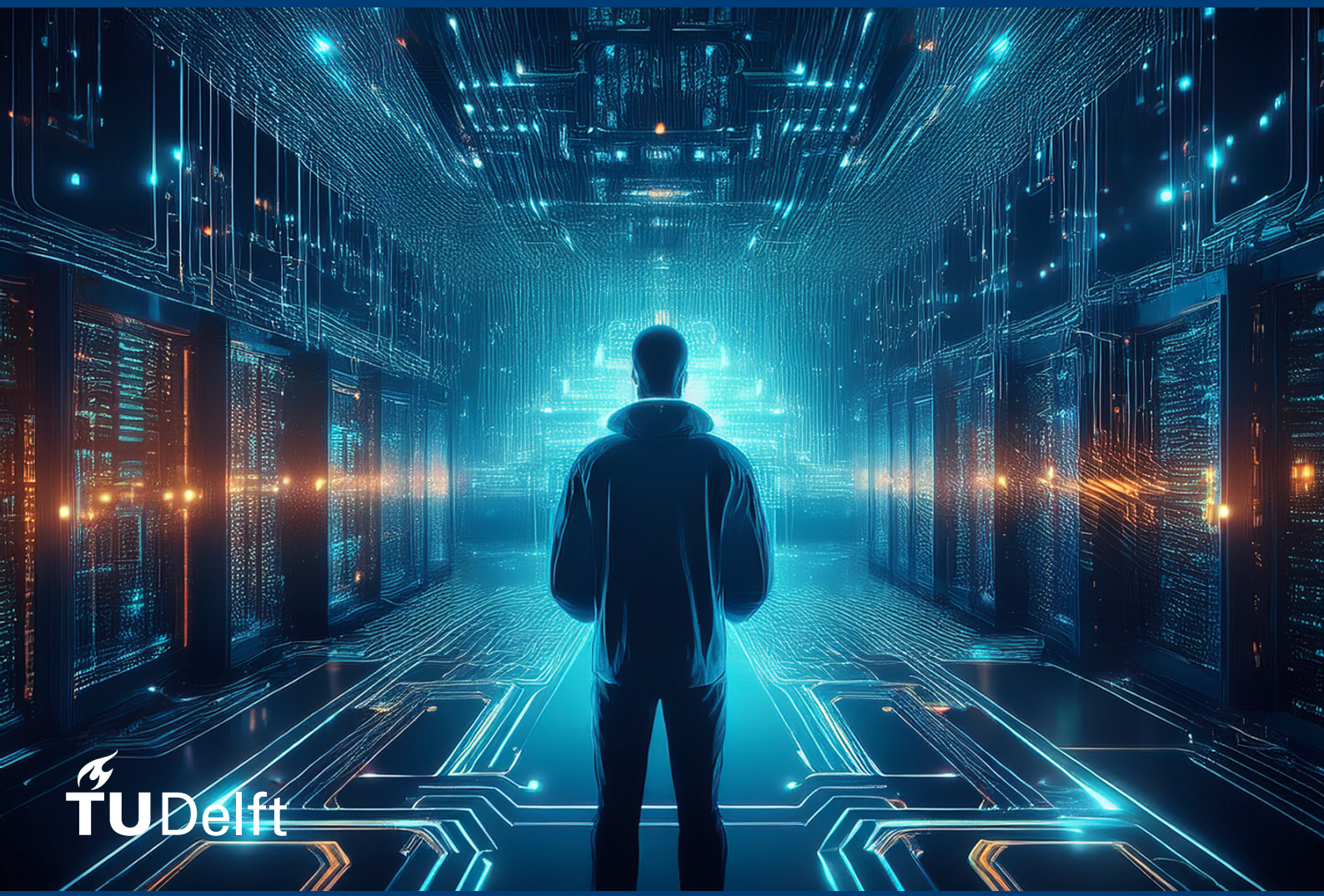
Beyond Real Traffic

Assessing the Reliability of AI-Generated Network Data in Deep Learning-Based Intrusion Detection Models

H.E.J. Bosma, BSc

December 7, 2025



Beyond Real Traffic

Assessing the Reliability of AI-Generated Network Data in Deep Learning-Based Intrusion Detection Models

by

H.E.J. Bosma, BSc

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday December 16, 2025 at 13:30.

Student number: 4536215
Project duration: August 15, 2024 – December 16th, 2025
Thesis committee:

Prof. dr. G. Smaragdakis,	TU Delft, thesis advisor
Dr. K. Liang,	TU Delft, supervisor
Dr. Ir. J. Dauwels,	TU Delft

TU Delft

Preface

At the start of this journey, I started with a different topic than I ended up with. The proposed topic was aimed at improving existing IDSs by taking application data into account and somehow combine this into a general trust score. For this I needed datasets that proved to be old, cumbersome and sometimes difficult to find. It was on a day that my dear colleague Ton came up with one remark that changed it all. ”What if you let ChatGPT create you a dataset, like writing a thousand emails”. That inspired me to explore the use of generated data in IDS. I would like to thank my supervisor dr. Katai Liang for exploring this topic with me and giving me the freedom to shape this into a thesis.

Then there is dr. Rui Wang, who I’ve spent many meetings with discussing the approach of the measurements, possible solutions to problems that occurred on the way and interpreting the results. He went all the way even making himself available during his free time to help me whenever I needed. Without you, this thesis would not have come the way it did. Thank you Rui! And remember, you are still very welcome to visit at the other side of the campus and check out the labs of Electrical Engineering!

Of course this thesis would also not have been possible with the help of some of my family, friends and colleagues. I would like to thank my parents and my colleagues and friends: Ton, Peter and Mahdi for hosting measurement stations over at their houses for me to collect the material that I needed. Ton in particular, for bringing up a small bug in the station that did not allow the software to run. After spending an evening debugging this over text messages, the measurements could really take off. Of course I want to thank all my amazing colleagues of the EEE group, for giving me the motivation to keep going when things got difficult and providing me with the space and time to bring this thesis to a success. Thank you Lenny, in particular, for arranging the defence of this thesis. I wouldn’t know where to have started.

Then I want to thank my best friend Ege, for aiding me with LaTeX, that still proves to be more of a challenge than sometimes initially thought. In general Ege has also inspired and motivated me by bringing up ideas and going through brainstorm sessions. Thank you dear friend!

Then there are my housemates, that gave me the space and support at home for me to be able to finish the writing. And being able to help me out at moment’s notice when needed. Thank you all!

Last but not least, I want to thank Tamsyn, my girlfriend, for supporting and motivating me through this journey. And for the extensive proof-read that she performed right up until the deadline. I don’t know where I would have been without you!

This masters’ programme was an amazing journey, where I’ve learned more than just what the courses provided. But after many years of study, it is time that my role as a student at this university comes to an end. But ties will not be cut. Now my time has arrived to pass on my experience and knowledge from the other side of the classroom, to students that start freshly with their journey! I can only wish them the same amazing years that I went through myself during this phase of life.

H.E.J. Bosma, BSc
Delft, December 2025

Contents

List of Figures

List of Tables

Abbreviations

Abbreviations

Abbreviation	Full form
AM	Application Model
ANN	Artificial Neural Network
BPTT	Back Propagation Through Time
CNN	Convolutional Neural Network
DOS	Denial of Service
IDS	Intrusion Detection System
GAN	Generative Adversarial Network
LLM	Large Language Model
LSTM	Long-Short-Term Memory
NTM	Network Traffic Model
NTT	Network Traffic Telemetry
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
VM	Voting Model
WD	Wasserstein Distance

1 Introduction

In the world of machine learning and deep learning, one factor plays a decisive role in distinguishing between successful models and wasted effort: the dataset used for training and testing. As deep learning models have become increasingly complex over the years, with rapidly growing numbers of nodes and weights, there has been an accompanying demand for ever larger and more diverse datasets. The introduction of large-scale models such as OpenAI's ChatGPT [21] and DeepSeek's Large Language Model (LLM) [7] exemplifies this trend.
In the case of Artificial Neural Networks (ANNs), a commonly cited heuristic is that the amount of data required should be at least 50 times greater than the number of adjustable parameters in the model [3]. Obtaining such vast amounts of data can be both costly and time-consuming. Depending on the specific application, datasets may need to be purchased, collected using specialized hardware, or generated through controlled experiments. Furthermore, when time-sensitive or domain-specific data is required, the collection process becomes even more challenging.

1.1. Intrusion Detection Systems (IDS)

Network Intrusion Detection Systems are designed to monitor network traffic and identify anomalous or malicious activity. Their primary goal is to detect and alert on traffic patterns that deviate from established definitions of normal behaviour. Broadly, IDS approaches can be divided into two categories:

- **Rule-based Intrusion Detection**, where incoming traffic is evaluated against a predefined set of signatures or rules [17];
- **Deep Learning-based Intrusion Detection**, where anomalies are identified through models trained on large datasets of network behaviour [18].

1.2. Data Challenges in Deep Learning-based IDS

Deep learning-based IDS models, while powerful, are highly dependent on the quality and diversity of the datasets they are trained on [37]. In cybersecurity, obtaining representative and up-to-date labelled data is particularly difficult due to privacy constraints, the dynamic nature of network environments, and the scarcity of publicly available datasets. Well-known benchmarks such as NSL-KDD and CICIDS2017 are frequently reused, but they may not reflect current attack trends or traffic distributions [39]. The consequence of this, is that more recently developed methods to intrude a network could bypass the IDS that were trained with these older datasets. Or, with the technology behind this constantly being updated, it could yield more false negatives. What means that perfectly fine network traffic would be labelled as anomalous.

As a result, researchers are increasingly exploring the use of AI-generated network data to supplement real datasets. Generative models, such as Generative Adversarial Networks (GANs) and diffusion-based models, can create synthetic network traffic that mimics real-world distributions [1]. This approach offers advantages in data availability, cost, and privacy preservation.

However, questions remain about how reliably such data can be used to train intrusion detection models and whether it can generalize to real network conditions. Another approach could be the use of LLMs or other transformer-based models for the generation of training data. Although it is currently already being researched if LLMs could play an active role as a base for IDS [9], it appears that the role of an LLM to act as data generator for training other deep learning-based IDS has not yet been systematically investigated as of the day of this writing.

1.3. Problem Definition

While AI-generated network traffic provides a promising solution to data scarcity in cybersecurity research, the reliability of such synthetic data, and its broader applicability across other data modalities, remains largely unexplored. In order to increase the accuracy of IDS, it is sometimes useful to include data belonging to the application of the system into the IDS [16]. Obtaining representative and high-quality application layer datasets, however, poses additional challenges. These datasets are often context specific, hardware dependent, and expensive to collect. In such cases, AI-generated data may offer a powerful mechanism to augment or replicate measurements from real-world systems, reducing costs and increasing dataset diversity. In order to use this way of collecting datasets properly however, some topics need to be understood more thoroughly:

1. To what extent AI-generated data can supplement or replace real-world data (both network and application-level) in training deep learning-based IDS models;
2. Whether synthetic data can accurately capture the temporal and contextual dependencies present in real sensor measurements;
3. How the ratio of real-to-synthetic data affects model performance.

The central problem addressed in this thesis is therefore the **uncertainty surrounding the reliability and performance impact of AI-generated data, encompassing both network- and application-level information, when used to train deep learning-based IDSs**. This thesis aims to determine the extent to which AI-generated samples can be used to complement or replace real-world data without significantly reducing model performance.

To create this thorough understanding of these questions, this thesis will describe a set of experiments where the performance of Deep Learning-based IDS will be evaluated compared to the datasets that was fed into their training. Specifically the following steps will be taken to retrieve answers to the previous sets of questions:

1. To start, one needs to look at Deep Learning-based IDSs in general and the role of the training data that is being used:
   (a) The baseline performance of Deep Learning-based IDS needs to be established with different configurations;
   (b) The accuracy of IDS trained with real Network Traffic Telemetry (NTT) data and with Application Model (AM) data added is to be compared.
2. After establishing the optimal base configuration for the IDS, the data itself should be examined more in depth and the ability of Generative AI to replicate this should be explored:
   (a) What input should be given to Generative AI to realise a usable dataset;
   (b) How does the generated data compare to a real dataset in terms of dataset statistics.
3. After obtaining both real- and generated datasets, these can be used to train and evaluate the performance of IDS:
   (a) A data subset should be extracted to evaluate the created models on accuracy;
   (b) A set of models should be trained, each with a different ratio of real and generated data in it's training dataset and its accuracy evaluated.

1.4. Thesis Outline

This thesis started with introducing the problem of datasets in training deep learning-based IDS. In Chapter 2, relevant background information on IDS in general, deep learning-based IDS, the Long-Short-Term Memory (LSTM) model and the Wasserstein distance (WD) will be described. In Chapter 3, a small application context for this thesis will be introduced, together with the architecture of a baseline IDS. There will be an analysis performed on the recorded application- and NTT data, after which artificially generated data will be introduced and analysed. Two baseline models will be trained to serve as a benchmark for the experiments in Chapter 4. In this Chapter, two sets of experiments will be stated and the experimental results will be demonstrated. In Chapter 5, the results will be discussed and explanations will be provided, after which a conclusion is drawn. And last, the thesis will be summarised and concluded in Chapter 6.

2

Background

The reliability and effectiveness of IDS depend not only on the sophistication of their algorithms but also on the quality and representativeness of the data on which they are trained. In recent years, deep learning models have increasingly been applied to IDS tasks, offering improved adaptability and the ability to recognize complex temporal and statistical patterns in network and application traffic [18]. Despite these advances, such models remain constrained by one persistent limitation: the scarcity of diverse and up-to-date datasets [39]. This limitation motivates the exploration of alternative data generation methods, including the use of artificial intelligence (AI) models capable of producing realistic synthetic samples.

This chapter provides the conceptual and technical foundation necessary to understand the research presented in this thesis. It begins by introducing the principles, architecture, and classification of IDS, distinguishing between rule-based and data-driven approaches. The discussion then moves toward the integration of deep learning techniques in IDS, summarizing the progress achieved and the challenges that remain, particularly in relation to data availability and generalization. Subsequently, attention is directed to neural network architectures relevant to this work, with a specific focus on LSTM-Recurrent Neural Networks (RNNs). These models are especially suitable for analysing sequential and time-dependent data, such as network flows or application-layer sensor readings, and form the basis of the intrusion detection framework developed in this study. Afterwards, the chapter explores the emerging potential of LLMs as generative tools for creating synthetic datasets. Last but not least, the chapter will introduce the WD and discuss how this tool possibly can be used as an estimator for how much real recorded data would still need to be used in combination with the LLM-generated data.

2.1. IDSs

IDSs play a central role in modern cybersecurity by continuously monitoring network activities to identify anomalies that may indicate malicious or abnormal behaviour. Their purpose is to detect, and in some cases respond to, unauthorized access, data exfiltration, or disruption attempts within digital infrastructures. IDSs are typically deployed at various layers of a system ranging from network gateways to end hosts. There they form a great addition to existing defence mechanisms like firewalls and encryption by recognising ongoing attacks or policy violations and reacting to this. IDS approaches can broadly be classified into the two main categories mentioned in Section 1.1.

2.1.1. Rule-based IDS

Rule-based IDSs rely on predefined signatures, heuristics, or expert-defined rules that describe known attack patterns. They operate similarly to antivirus software, comparing observed traffic or system events against a set of rules that is built based on known threats. This approach is effective in detecting well-understood and previously documented attacks but is limited by its inability to recognize new or evolving threats that do not match existing patterns [19]. Examples of rule-based IDS include widely used tools such as Snort and Suricata, which rely on continuously updated rule sets to maintain detection accuracy.

2.1.2. Anomaly-based IDS

Anomaly-based IDSs, in contrast, learn a model of what constitutes normal system or network behaviour and flags any significant deviations as potential intrusions. Early implementations of anomaly detection relied on statistical models or traditional machine learning algorithms; more recently, deep learning techniques have been introduced to improve their ability to model complex and high-dimensional data [37]. This approach enables the detection of previously unseen attacks, including zero-day exploits, but it also introduces challenges such as higher false positive rates and the need for large, representative training datasets. In practice, modern cybersecurity architectures often integrate both rule-based and anomaly-based detection in a hybrid IDS configuration. Such systems leverage the precision of signature matching for known attacks while using anomaly detection to capture novel threats and behavioural irregularities [39]. The following Section explores how deep learning methods have been used to implement modern-day anomaly-based IDS.

2.1.3. Deep Learning-based IDS

Deep learning has revolutionized the field of intrusion detection by enabling models to automatically extract hierarchical features from large, high-dimensional datasets. Unlike traditional machine learning techniques, which rely heavily on manual feature engineering, deep learning architectures can capture non-linear and temporal dependencies directly from raw network traffic or application data [11]. This capability allows for more generalizable and adaptive IDSs that can identify complex or previously unseen attack patterns.

Several deep learning paradigms have been successfully applied to IDS research, including Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), RNNs, and hybrid models that combine multiple architectures. Ferrag et al. [11] provide an extensive comparative study of these approaches across different datasets such as CSE-CIC-IDS2018 and Bot-IoT, highlighting that deep discriminative models (e.g., RNNs, DNNs, CNNs) generally outperform traditional machine learning methods like Support Vector Machines (SVMs) and Random Forests. However, they also emphasize the sensitivity of these models to dataset characteristics, particularly class imbalance and representativeness, which significantly affect their generalization to real-world network environments.
Recurrent architectures, especially LSTM) and Gated Recurrent Unit (GRU) networks, have shown notable promise due to their ability to model sequential and temporal dependencies in network traffic [38, 32]. These models maintain internal states that allow them to remember contextual information across time steps, making them particularly suited for IDS tasks where attack behaviour unfolds over time. This in contrast to Feed Forward only DNNs like CNNs that only feed the data foward to the next layer and do not take into account previous inputs. Yin et al. [38] demonstrated that LSTM-based models trained on the NSL-KDD dataset achieved higher accuracy and lower false positive rates than classical RNNs, validating their effectiveness in capturing long-term temporal patterns. Similarly, Tang et al. [32] showed that GRU-based models could achieve strong detection rates with reduced computational complexity, highlighting their efficiency for real-time applications.

2.2. LSTM RNN

As is mentioned in Section 2.1.3, LSTM networks are very suitable to be used in anomaly detection based IDS due to their ability to maintain an internal state based on previous inputs and therefore process time dependant data in a more effective way.

LSTMs are a special category of RNNs. These have originally been created to mitigate an issue that RNN's tempt to show when a long sequence of data, e.g. speech or weather models are being used [14]. In general with RNNs, data is only fed back into the same node or perhaps a few nodes back. However, when a long sequence of data is fed into the model, the influence of data entries from relatively long ago (e.g. more than 10-20 steps back in time) is not being used any more due to the vanishing gradient problem [14]. The LSTM addresses this issue by using a memory cell that keeps track of older inputs and can be updated with fresh input. Gers et al. [12] added a so-called forget gate to the LSTM that can reset the memory cell in order to keep training practical and the model agile enough to adjust to shifts in data streams.

2.2.1. Inner Working of the LSTM Cell

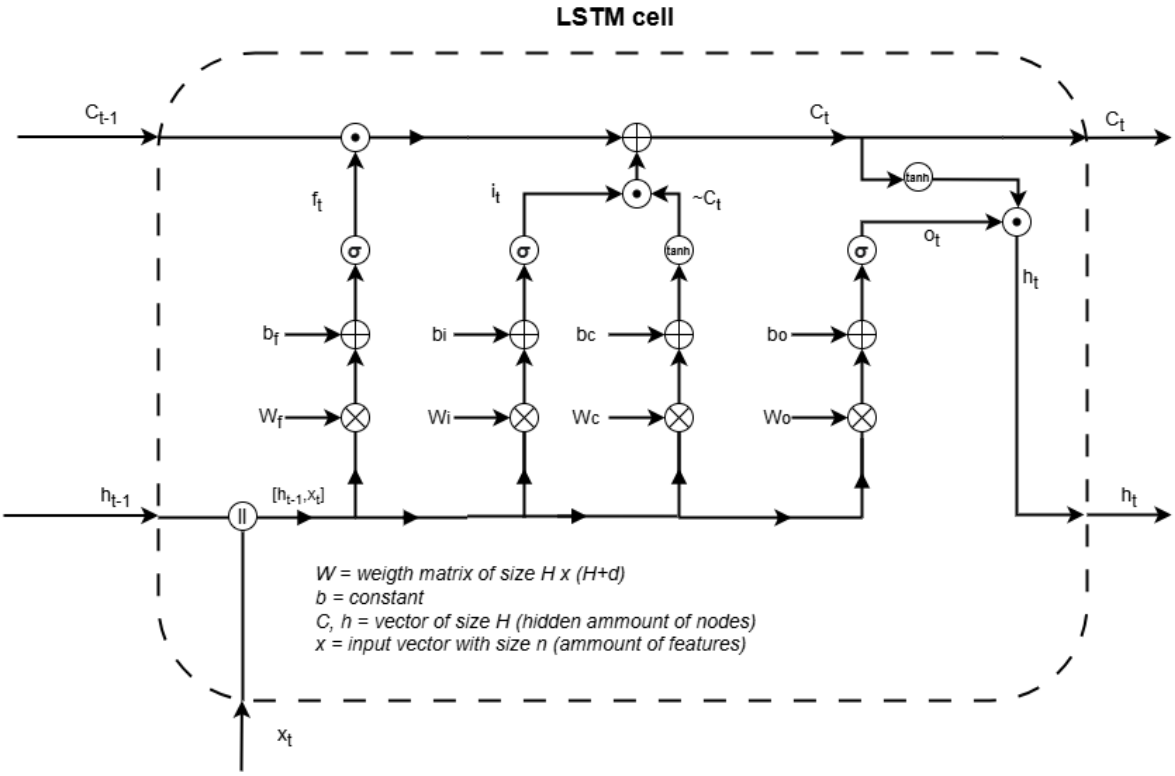


Figure 2.1: Schematic view of an LSTM cell. Short term memory is stored in the hidden nodes vector h_t while longer term memory is stored in vector C_t. The output is obtained by taking the vector h_t and combining all the nodes into the desired output shape.

The Long Short-Term Memory (LSTM) cell extends the conventional recurrent neuron by introducing a dedicated internal *memory cell* C_t and three multiplicative *gates*. These gates regulate the flow of information into, within, and out of the cell, enabling the network to maintain, overwrite, or discard information selectively. This architecture allows the LSTM to preserve relevant temporal dependencies across long sequences while mitigating the vanishing gradient problem inherent to classical RNNs [14].

As illustrated in Figure2.1, each LSTM cell operates on three primary inputs: The current input vector x_t, (representing the input features at time step t), the previous hidden state h_{t-1} (which carries short-term contextual information from the previous step) and the previous cell state C_{t-1} (which functions as the long-term memory).

From these inputs, the LSTM computes three gating functions—each implemented using a sigmoid activation—and a candidate cell update vector, computed using a hyperbolic tangent activation. These components are defined as:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad \text{(Forget gate)} \tag{2.1}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad \text{(Input gate)} \tag{2.2}$$

$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \quad \text{(Candidate cell update)} \tag{2.3}$$

Here:

- W_f, W_i, W_C are the weight matrices corresponding to the forget, input, and candidate update gates respectively,
- b_f, b_i, b_C are the associated bias terms,
- $\sigma(\cdot)$ denotes the sigmoid activation function, which outputs values in the range $(0, 1)$,

- $\tanh(\cdot)$ denotes the hyperbolic tangent activation, which outputs values in the range $(-1, 1)$,
- The concatenation $[h_{t-1}, x_t]$ represents the combination of the previous hidden state and the current input.

The forget gate f_t determines the proportion of the previous cell state C_{t-1} that is retained, effectively "forgetting" irrelevant information. The input gate i_t controls how much new information from the candidate update $\tilde{C}_t$ is written into the memory. The new cell state C_t is then computed as:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{2.4}$$

Where $\odot$ denotes element-wise multiplication. The first term in Equation 2.4 controls the persistence of prior memory, while the second term introduces newly computed information. This additive update mechanism, as opposed to purely multiplicative recurrence in standard RNNs, allows for a more stable gradient flow through time, thereby enabling learning over long sequences. Next, the cell computes the output gate and the updated hidden state:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad \text{(Output gate)} \tag{2.5}$$
$$h_t = o_t \odot \tanh(C_t) \quad \text{(Updated hidden state)} \tag{2.6}$$

Where:

- W_o and b_o are the weight matrix and bias for the output gate,
- o_t determines how much of the current cell state contributes to the output,
- h_t is the updated hidden state that serves both as the immediate output and as input to the next time step.

As shown by [12], the introduction of the forget gate enhances this architecture by enabling the cell to explicitly reset or decay its memory, preventing uncontrolled accumulation of information and allowing the network to adapt efficiently to non-stationary input sequences. This capability has made LSTMs particularly powerful in tasks involving temporal dependencies, such as natural language modelling, speech recognition, and time-series-based IDSs. A schematic overview of these interactions, including the gate activations, cell state transitions, and hidden state outputs, is depicted in Figure 2.1. The diagram directly corresponds to the mathematical formulation presented in Equations 2.1–2.6.

2.2.2. Training of LSTM Networks

Training an LSTM network follows the general principles of training RNNs, but with specific adaptations to accommodate its gating mechanisms and cell state dynamics. The objective is to optimize the trainable parameters

$$\Theta = \{W_f, W_i, W_C, W_o, b_f, b_i, b_C, b_o\}$$

such that the network minimizes a loss function $\mathcal{L}$, typically defined over a sequence of predictions compared to the true outputs. To train these parameters, the LSTM uses a variation of the Back Propagation Through Time (BPTT) algorithm. In BPTT, the network is unfolded across time steps, creating a feed-forward computational graph where each unfolded cell corresponds to one time step. The gradients of the loss function with respect to the parameters are then computed by applying the chain rule backward through this temporal structure.

The parameter updates are performed using gradient-based optimization methods such as Stochastic Gradient Descent (SGD) or its variants (e.g., Adam or RMSprop):

$$\Theta \leftarrow \Theta - \eta \frac{\partial \mathcal{L}}{\partial \Theta} \tag{2.7}$$

where η is the learning rate.
The key difference between standard RNNs and LSTMs lies in how gradients propagate through time. In a traditional RNN, repeated multiplications by the recurrent weight matrix W_{hh} cause gradients to

either diminish exponentially (*vanishing gradient*) or grow uncontrollably (exploding gradient) as sequence length increases [5]. This limits the ability of conventional RNNs to capture dependencies beyond approximately 10–20 time steps. In contrast, the LSTM's additive cell state update (Equation 2.4) provides a nearly constant error flow, often referred to as the Constant Error Carousel [14]. The gradient with respect to the cell state can be expressed as:

$$\frac{\partial \mathcal{L}}{\partial C_{t-1}} = \frac{\partial \mathcal{L}}{\partial C_t} \cdot f_t \tag{2.8}$$

where the forget gate f_t directly scales how much of the gradient is propagated backward. Since this multiplication can be controlled through the gating function, the LSTM maintains stable gradients across long sequences and effectively learns long-term dependencies. During training, regularization techniques such as dropout and gradient clipping are often employed to prevent over fitting and suppress residual gradient explosions. The resulting network is capable of robust temporal modelling even in the presence of noisy or non-stationary input data, making it highly suitable for IDSs based on time-dependent behavioural patterns.

2.2.3. LSTM as IDS

The long memory aspect of the LSTM RNN makes it very suitable to be trained upon a time sequence of data, like the data type can be found in NTT data upon the arrival of network packages onto a server, and then serve as an IDS. This is exactly what has been done by Jiang et al.[16]. In this research, a multi-channel IDS had been created, where application data and network telemetric data were used both separately and together to train 3 different LSTM RNN's. Upon arrival of a packet, these would bring out a vote to a central voting counter. If the majority agreed upon accepting the node after receiving it's package, it would count towards a positive 'trust score'. If the voter's majority would agree on a denial, the node would be given a lower 'trust score'. Trained and tested with the NSL-KDD dataset, this would result in an accuracy upwards of 97%. It should be noted that with application data here is meant the features in the NSL-KDD dataset that are corresponding with application related network traffic e.g. 'is there a login attempt performed on the server'.

2.3. Data Challenges in Deep Learning-based IDS

The success of a Deep-Learning based IDS stands with the quality of the dataset that it has been trained on. Over the years, many datasets have become available for this purpose [11]. These datasets very from Network traffic-based datasets e.g. KDD Cup 1999 or Darpa 1998 to Internet traffic-based datasets e.g. CAIDA internet traces 2016. For this thesis, the focus will lay on the NSL-KDD dataset [33]. This dataset, as was used as well by the proposed LSTM-based model in Section 2.2.3, is an improved version of the KDD Cup 1999 dataset. Duplicate and redundant records had been removed and the number of records is more reasonable. This dataset is a Network traffic-based dataset. Features include, but are not limited to: The protocol type, the amount of source bytes, the count of sent messages in the last 30 seconds, etc. A more complete overview will be given later in Chapter 3

This dataset as well as others have one trait in common. They are, in general, dated and difficult to collect. The NSL-KDD dataset is based on the 1999 KDDcup dataset [36], that was 'created for a classifier learning contest to learn a predictive model capable of distinguishing between legitimate and illegitimate connections in a computer network' [31]. Over the years, a lot of the networking infrastructure and technology as it was in 1999 is still in use. However, as new technologies emerged and systems have been upgraded to newer architectures over time with better security features, these older datasets might not be as much reflecting the current set of threats that exist to computer networks any more as they did in 1999. Now newer datasets are available [11]. However, with the continuous struggle between developers of secure systems and networks and undiscovered bugs in various systems or even malicious entities trying to circumvent these measures, it is just a matter of time for these datasets to lose their relevance in the progress over time. And constant updating of available datasets will remain needed.

Collecting these datasets create difficulties on their own. Often they rely on simulations [24], because collecting real world network traffic is generally infeasible due to privacy regulations, ethical constraints

or legal implications of exposing sensitive user data. This not only applies for network traffic data, but even more so for application data. On top of this, real-world measurements are often expensive and need specific hardware. They can also take up a lot of time where a solution for an application specific IDS would be needed within shorter time constraints.

2.4. Generating Data with LLMs

2.4.1. Generative AI

Generative artificial intelligence (AI) refers to models that learn an approximate data distribution and can sample new, synthetic observations from it. This includes families such as Generative Adversarial Networks (GANs) [13], and more recent diffusion-based models for high-quality image synthesis [25]. Specialised generative systems now exist for images, audio, and video, while more general-purpose models can produce text and structured data.

For deep learning-based IDS, generative AI offers a potential remedy to data scarcity: once a model has learned a suitable approximation of the underlying distribution of network or application-layer data, it could be used to augment limited real measurements with additional synthetic samples. This can increase the effective training set size and help expose the model to a broader range of patterns, without requiring proportionally more measurement campaigns or specialised hardware. Synthetic data is therefore best viewed as a complement to empirical datasets, particularly in scenarios where controlled experiments or long-term monitoring are difficult to perform.

However, the use of generative models for dataset construction also introduces new risks. Synthetic data inherits the biases and blind spots of the training distribution and of the model architecture itself. Recent work by Shumailov et al. [29] shows that repeatedly training models on their own generated outputs can lead to 'model collapse', where the model progressively forgets low-probability regions of the original data distribution and converges to increasingly distorted, low-diversity outputs. However, results of research by Alemohammad et al. [2] indicate that mixing in a non-trivial fraction of real data at each training generation can substantially mitigate this degradation and preserve more faithful sampling behaviour.

These findings suggest that generative AI is a powerful but delicate tool for data generation in IDS research. Synthetic samples can help alleviate the cost and difficulty of collecting real-world network and application-layer data, but they should not fully replace empirical measurements. Maintaining a continuous stream of real data is essential to anchor training and prevent distributional drift caused by recursive use of synthetic outputs. Within this broader landscape, LLMs are particularly interesting due to their versatility and ease of use through prompting; their role in generating structured, security-relevant data will be examined in more detail in Section 2.4.2.

2.4.2. LLMs

LLMs such as ChatGPT [21] and DeepSeek [7] generate text by predicting sequences of tokens based on patterns learned from vast corpora of natural language. Most state-of-the-art LLMs are based on the Transformer architecture introduced by Vaswani et al. [35], which relies on self-attention mechanisms to model long-range dependencies within sequences. During training, the model learns statistical relationships between tokens in diverse contexts, enabling it to produce coherent, structured, and contextually appropriate text when prompted.

Although originally developed for natural language processing tasks, LLMs are increasingly being applied to domains involving structured or semi-structured data [30]. Their ability to understand prompts, generate consistent sequences, and follow formatting instructions makes them suitable for synthesising time-series data, structured logs, or domain-specific records. For the purpose of this research, this versatility is especially relevant: application-layer sensor data and network metadata often follow predictable temporal and semantic patterns, which LLMs can reproduce when given suitable conditioning through prompting or in-context examples. As a result, LLMs offer a high-level, accessible means of generating synthetic datasets without requiring specialised generative models such as GANs or diffusion models.

However, the use of LLMs for data generation also introduces significant risks. First, the generated data will inevitably reflect the biases or blind spots present in the model's training corpus. If the model has not been exposed to patterns relevant for intrusion detection, the resulting synthetic data may misrepresent rare but important behaviours. Second, LLMs tend to "hallucinate'': confidently producing plausible-looking but factually incorrect or statistically inconsistent sequences [15]. Without careful validation, such inaccuracies may degrade the performance of an IDS trained on these samples. On top of this, the earlier discussed model collapse problem shows the need to combine LLM-generated samples with sufficient real-world measurements to preserve distributional integrity.

2.4.3. Prompting

When using LLMs to generate synthetic data, the design of the prompt plays a crucial role in ensuring that the model produces structured, coherent, and domain-appropriate output. Delgado-Soto et al. [8] demonstrate this clearly in their work on generating synthetic network traffic using GPT-based models. Their findings provide a set of practical principles for designing prompts that elicit realistic protocol-conforming sequences rather than generic text.

1. First, prompts must explicitly encode the *structure* of the desired output. In the case of network traffic, Delgado-Soto et al. formulate prompts that specify packet fields, message order, and protocol semantics using structured templates rather than free-form descriptions. For example, they instruct the model to generate traffic in a line-delimited, key–value format corresponding to ARP, ICMP, DNS, or HTTP conversations. By constraining the output schema, they prevent the model from defaulting to natural language prose, which is its typical mode of generation. Their results show that such schema-based prompting significantly increases both syntactic validity and semantic plausibility of the generated traffic.

2. Second, the prompt should provide *contextual examples* illustrating correct domain behaviour. Delgado-Soto et al. use few-shot prompting in which one or more real packet sequences are provided as exemplars. These examples help the model learn the temporal ordering of messages, dependencies between fields, and typical conversational patterns (e.g., request–response dynamics in DNS or HTTP). They report that removing these examples leads to degraded protocol consistency and, in some cases, the model inventing fields or deviating from expected flow behaviour. This supports earlier findings in prompt engineering research that few-shot demonstrations increase adherence to domain-specific constraints.

3. Third, prompts benefit from including *explicit constraints* describing what the model must and must not do. In their work, Delgado-Soto et al. instruct the model not to create non-existent protocol fields, not to include explanatory text, and not to deviate from the given protocol's message sequence. These negative instructions serve as guardrails that steer the model away from common LLM failure modes such as hallucinated metadata, extraneous commentary, or mixing of protocols. Their evaluation highlights that such guardrails are essential for generating outputs that resemble valid packet traces rather than plausible-sounding but technically invalid text.

4. Fourth, the authors find it valuable to specify *semantic conditions* tied to the domain. For instance, when generating ICMP traffic, the prompt encodes protocol rules such as matching identifiers or appropriate type/code values. In HTTP traffic generation, they encode constraints such as ensuring that the server responds only after a client request. Embedding these domain-level invariants within the prompt enables the model to respect protocol logic even without explicit fine-tuning on packet-level data.

5. Finally, Delgado-Soto et al. emphasise the importance of *post-prompt validation*. Because LLMs can still produce inconsistent or erroneous sequences despite good prompting, they validate each generated flow using protocol parsers. This suggests that effective prompt engineering should be combined with automated validation or filtering stages when used for synthetic dataset generation.

Together, these findings indicate that LLM prompting for IDS-relevant synthetic data is most effective when prompts combine four core elements: (1) explicit structural templates, (2) example-based demonstrations, (3) clear constraints and guardrails, and (4) embedded domain knowledge. When applied

carefully, this strategy enables LLMs to generate structured outputs that approximate the behaviour of real network or application-layer data—an essential requirement for their use in training deep learning-based IDS models.

2.5. Expected Behaviour and Underlying Mechanisms

Altough LLM's are in general very complex models where the predictability of their outcome is difficult to estimate. Some predictions of the usability of the data can be made when it is mixed with real data. A principled way to estimate this is to treat the training data distribution as a mixture of real and synthetic distributions and analyse how divergence between this mixture and the true data distribution affects the expected test error. This approach follows classical statistical learning theory [20] and domain adaptation theory [4], which relate generalisation error to the divergence between training and test distributions.

2.5.1. Mixture Distribution Formulation

Let P_{real} denote the true underlying data distribution, as approximated by a real dataset such as in the case of this thesis application data like temperature measurements from a sensor, and let P_{synth} denote the distribution induced by LLM-generated synthetic samples. If a fraction $\alpha \in [0, 1]$ of the training data is real and the remainder is synthetic, then the effective training distribution is

$$P_{\text{train}} = \alpha\, P_{\text{real}} + (1 - \alpha)\, P_{\text{synth}}. \tag{2.9}$$

The test distribution is assumed to be the real distribution,

$$P_{\text{test}} = P_{\text{real}},$$

reflecting the operational environment in which the IDS will be deployed.

2.5.2. Generalisation Error Bound

Statistical learning theory establishes that the generalisation error on the real distribution can be bounded by the training error plus a divergence term [20]:

$$\text{Err}_{\text{test}}(h) \;\le\; \text{Err}_{\text{train}}(h) \;+\; D(P_{\text{train}} \,\|\, P_{\text{real}})\,, \tag{2.10}$$

Where $D(\cdot\|\cdot)$ is a statistical divergence. Domain adaptation theory provides a similar formulation, using the $\mathcal{H}\Delta\mathcal{H}$ divergence to quantify distribution shift [4]. In either case, the divergence between the training mixture and the real distribution directly influences the potential degradation of performance when the model is evaluated on real-world data. Substituting Equation 2.9 into the divergence term gives:

$$D\big(\alpha P_{\text{real}} + (1-\alpha) P_{\text{synth}} \;\big\|\; P_{\text{real}}\big)\,, \tag{2.11}$$

Which increases with both the deviation of synthetic samples from real data and the proportion of synthetic data used.

2.5.3. Divergence Approximation and Predictive Thresholds

For many divergence measures, the divergence between the mixture and the real distribution can be upper bounded by a linear scaling term involving the divergence between the synthetic and real distributions [34]:

$$D(P_{\text{train}} \,\|\, P_{\text{real}}) \;\le\; (1-\alpha)\, D\big(P_{\text{synth}} \,\|\, P_{\text{real}}\big)\,. \tag{2.12}$$

If a maximum tolerable divergence ϵ is defined—for example, the divergence that is similar to the divergence between two sets of real-measured data.—the inequality

$$(1-\alpha)\, D\big(P_{\text{synth}} \,\|\, P_{\text{real}}\big) \le \epsilon$$

yields the following estimate for the minimal amount of real data required:

$$\alpha_{\text{min}} \;\approx\; 1 - \frac{\epsilon}{D\big(P_{\text{synth}} \,\|\, P_{\text{real}}\big)}. \tag{2.13}$$

This expression provides a theoretically motivated threshold for the proportion of real data needed to ensure that synthetic samples do not introduce excessive distribution shift.

2.5.4. Practically Computing Divergence

Divergence can be estimated for numerical features by the so-called Wasserstein distance [22]. This can be used for example as follows:
Let $P_{\text{real}} = \{x_1, \dots, x_n\}$ and $P_{\text{synth}} = \{y_1, \dots, y_m\}$ denote two empirical sets of numerical application measurements (for example a temperature measurement), not necessarily of equal size. We model them as empirical distributions

$$P_{\text{real}} = \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}, \qquad P_{\text{synth}} = \frac{1}{m}\sum_{j=1}^{m} \delta_{y_j},$$

where δ_x denotes a unit point mass at x and n and m represent respective sample set sizes. In one dimension, and for the purposes of this work, we define the divergence

$$D\big(P_{\text{synth}} \,\|\, P_{\text{real}}\big) = \int_{-\infty}^{\infty} \big|F_{\text{synth}}(t) - F_{\text{real}}(t)\big|\, dt, \tag{2.14}$$

where F_{synth} and F_{real} denote the empirical cumulative distribution functions (CDFs) of P_{synth} and P_{real}, respectively. This choice of $D(\cdot\|\cdot)$ corresponds to the one-dimensional Wasserstein-1 distance (also known as the Earth Mover's Distance) between the two empirical distributions. Importantly, equation 2.14 does not require $n = m$ and naturally handles unequal sample sizes.

The mixture distribution framework provides a mathematically grounded approach for predicting the usability of LLM-generated data. If the divergence between synthetic and real samples is small, a large fraction of synthetic data can be safely incorporated. Conversely, high divergence implies that even small amounts of synthetic data may distort the learned decision boundaries of an IDS. The framework therefore offers both a theoretical and practical basis for deciding how much artificial data can be included in training deep learning-based IDS models.

3

Design

3.1. Context Setting

Up to now in this thesis, the general case of having an IDS including evaluating AM data as part of the anomaly detection has been discussed. In order to perform measurements on a system where the training data for this system can be artificially generated and compared with real-world data, one needs to implement a certain application to test this on.

3.1.1. Environmental Measurements

In this thesis, it is decided to take a small distributed network of environmental measurement stations as an application. These sensors, that will be placed over five different villages (Den Hoorn, Wassenaar, Dordrecht, Lage Zwaluwe and Delfgauw) in the Netherlands will collect the following data:

- **Temperature** [°C]: The temperature surrounding the weather station will be collected with an accuracy of 2 decimals;
- **Humidity** [%]: The relative air humidity will be collected with an accuracy of 1 decimal;
- **PM1,0** [$\mu g\,m^{-3}$]: A fine particle concentration of particles with a size of $1,0\mu m$ in the air;
- **PM2,5** [$\mu g\,m^{-3}$]: A fine particle concentration of particles with a size of $2,5\mu m$ in the air;
- **PM10,0** [$\mu g\,m^{-3}$]: A fine particle concentration of particles with a size of $10,0\mu m$ in the air.

These are collected by a central server that will make the data available in an overview for hosts of the weather stations. But to maintain integrity of the system and obtain correct measurements, one would include an IDS to prevent anomalous devices from either manipulating ongoing measurements by disruption, or inserting non-real data into the collection.

3.2. Environmental Measurement Stations

3.2.1. Hardware

The weather stations themselves consist of a Raspberry Pi (RPi) SoC computer [23]. These are either of the type: RPi 4B 4GB, RPi 4B 8GB or RPi 4B 8GB. Two sensors from the Seeed Studio collection [28] are connected to these RPis. The sensors and their specifications can be found in Table 3.1. These sensors are connected to the RPis using I2C to the header pins present on all RPi B models and also retrieve their power from the 5V power pins on the RPis. The RPi has by default one I2C bus enabled. In order to connect a second device to the RPi over I2C, a second bus has to be enabled and dedicated to some of the GPIO pins on the RPi. An example can be seen in Figure 3.1

3.2.2. Software

Each environmental measurement station operates on a Raspberry Pi running the ARM version of **Ubuntu 24.04.03 LTS (64-bit)** [6], which is installed through the Raspberry Pi Imager tool. On top of

Table 3.1: Specifications of the Grove Sensors Used

Specification	**Grove – Laser PM2.5 Sensor (HM3301) [26]**	**Grove – Temperature & Humidity Sensor (DHT20) [27]**
Operating voltage	3.3 V / 5.0 V DC	2.0 V / 5.5 V DC
Operating temperature	$-10\,^{\circ}\mathrm{C}$ to $60\,^{\circ}\mathrm{C}$	$-40\,^{\circ}\mathrm{C}$ to $+80\,^{\circ}\mathrm{C}$
Operating humidity	10 % – 90 % RH (non-condensing)	0 % – 100 % RH
Measurement type	Particle concentration ($PM_{1.0}$, $PM_{2.5}$, PM_{10})	Temperature and humidity
Effective PM2.5 range	1–500 $\mu g\,m^{-3}$	–
Maximum concentration range	1000 $\mu g\,m^{-3}$	–
Resolution	1 $\mu g\,m^{-3}$	–
Stability time	~30 s after power-on	~1 s after power-on
Humidity accuracy	–	± 3 % RH (at $25\,^{\circ}\mathrm{C}$)
Temperature accuracy	–	$\pm 0.5\,^{\circ}\mathrm{C}$
Interface	I2C (address: 0x40)	I2C (address: 0x38)

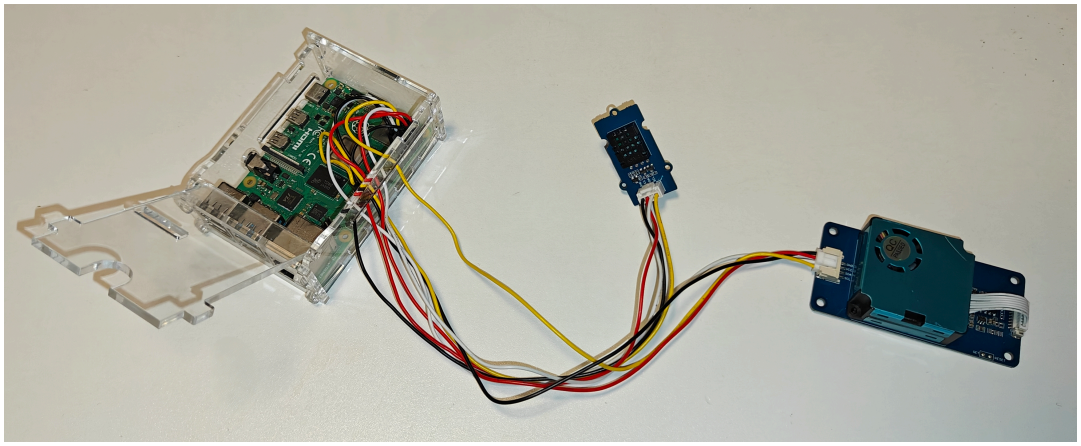


Figure 3.1: RPi 4B 8GB with Grove sensors connected.

the operating system, two custom software services have been developed specifically for this thesis. These services run continuously in the background and together provide both local sensor access for nearby users and remote data transmission to the central server infrastructure.

Local Sensor Access via Web Dashboard

The first service is a Python `Flask` web application that exposes a local dashboard for reading the real-time sensor values of the measurement station. This interface is intended for hosts or maintainers of the weather station who visit the device physically and connect their laptop directly to the RPi. To support such a direct connection, each station runs a lightweight DHCP service, assigning an IP address to any directly connected client machine. To improve usability, a minimal DNS service is also deployed. The Flask application retrieves sensor data from the I2C devices and presents the sensor measurement values on a simple HTML dashboard, enabling quick diagnostics during installation or maintenance.

Automated Data Acquisition and Uplink

The second background service is a Python script responsible for performing periodic sensor measurements and transmitting them to the central data collection server. The measurement interval is fully configurable through the service configuration file, allowing sampling frequencies to be adapted to experimental requirements or hardware limitations.

Each transmitted data packet includes the timestamp of the measurement, the raw sensor outputs (temperature, humidity, PM1.0, PM2.5, PM10), the geographical or logical location of the measurement station (set statically in the service configuration), a station-specific UUID that uniquely identifies the device.

For the purpose of training both the network-traffic and application-level machine-learning models, an additional field is included: a label indicating the operational status of the station at the time of measurement. This label is injected by the service and is not derived from the sensors themselves. The semantics of the labels are as follows:

- **0** — All sensors functioning correctly; normal behaviour.
- **1** — The station transmits data without rate-limiting, simulating a mild Denial-of-Service (DoS) pattern originating from the device itself, which should be considered anomalous network behaviour.
- **2** — One or more sensors intentionally producing abnormal or faulty values, enabling the study of sensor-level anomalies.

These labels are used solely during training and evaluation phases to support supervised learning and anomaly-detection benchmarking, but should not be considered part of a normal application set-up.

3.3. Data Collection

3.3.1. Server

The central component of the data-collection pipeline is a dedicated server that receives, processes, and stores the measurements transmitted by all environmental stations. For this research, a refurbished consumer-grade laptop was repurposed as the server system. Despite its modest hardware, it provides more than sufficient computational capacity for handling the incoming sensor streams, running network-monitoring tools, and executing the data-processing scripts required for the intrusion-detection experiments.

Server Hardware and Operating System

The server machine is a **Toshiba Satellite C50-A-1G6** laptop equipped with an Intel® Core™ i3-3110M CPU and 8,GB of RAM. It runs the AMD64 version of **Ubuntu 24.04 LTS**, ensuring compatibility with the full software stack used in this thesis, including the Flask-based collection service, Zeek, and the subsequent data-processing pipeline.

Data Collection Service

Running on the server is a Python-based back-end service responsible for receiving the periodic uploads from all environmental measurement stations. This service exposes an HTTP endpoint at `/upload`, to which each RPi sends a JSON payload containing containing the previously mentioned dataset. Upon receiving a request, the server performs the following operations:

1. Associates the measurement with metadata such as the client IP and port,
2. Queries the most recent Zeek network-traffic observations for the corresponding device,
3. Computes a set of NSL-KDD–inspired network-flow features,
4. Appends both AM and NTT data into two CSV log files:
    - A full log merging Zeek features + AM data,
    - An application-only log for analysing sensor behaviour.

The service continuously maintains these CSV files and makes them available to the analysis pipeline described in later chapters. A password-protected dashboard is also hosted on the server, enabling real-time inspection of the incoming AM data across different measurement locations. This was made both to check for the functionality of measurement stations and was made available to hosts of the measurement devices as an incentive to host such a device as well.

3.3.2. Measurements

Weather stations were spread across the Netherlands over five different locations as was mentioned in Section 3.1.1 to ensure that the weather data it measures is variable enough for the deep learning model to be trained with. However, it turned out that due to internet connectivity issues, the measurement station in Delfgauw failed to send collected data over to the server. Therefore, from now on, this thesis will continue with the data from just 4 of the nodes.

The application was configured to take a measurement every **5 minutes** and send the data as mentioned in Section 3.2.2 automatically to the server. As a start time, the client would send a first sample immediately on startup of the application and then continue from there. Therefore the measurements did not arrive synchronously to the server. Measurements were recorded for the period of a week in between the 31st of October 2025 at 20:30 and the 7th of November at 21:00.

Table 3.2: Total amount of estimated measurements versus the actual result.

Nr. of Stations (N):	**Total Recording Time** (T) [min]:	**Sampling Frequency per Station** (F_s)[Samples/min]:	**Estimated Total Samples** ($N * T * F_s$):	**Actual Sample Set Size**:
4	10,110	0.25	10,110	9279

As can be seen in Table 3.2, the estimation of total amount of samples deviates with 831 samples. This is due to the fact that a sensor did not always manage to send a sample exactly every five minutes. This is mostly due to connectivity issues of the weather stations.

3.3.3. Anomaly Generation

In order to train the Deep Learning model for the AM data properly, some 'anomalous' data had to be created as well. The two types of anomalous data are simulations of faulty sensors or malicious use of the server. To be exact about these events that took place, an overview can be found in Table 3.3. All of these events were performed by the weather station that was located in *Den Hoorn*. This due to the reachability of the device and the nature of one of the events that took place. This event involved sending a large amount of post requests to the server in a very short time. To avoid misuse of public infrastructure it was decided to perform this on the local network only.
Anomalous data has only been generated through the API of the application. This only allowed HTTP traffic to be recorded. Therefore other famous types of attacks as are represented in e.g. the NSL-KDD dataset, are not present in these measurements.

3.3.4. Data Analysis & Preparation

In order to compare the data that will be generated by an LLM to the measured data, it is important to know the statistics related to the dataset. Since for both the AM data and the NTT data, synthetic data will be generated, for both types of data some statistics will be given. Most important here will be the WD (see Section 2.5.4). Since this will indicate how comparable the data is that will be generated to the original measured data.

AM data
The features collected for the AM are the sensor values that were described in Section 3.1.1. The anomaly to normal data ratio is quite low as can be seen in Table 3.3. Therefore, additional anomalies were added by transforming some of the good data into anomalous data. This was done by replacing 500 regular good samples with samples that had their pm measurements set to 0 or their temperature and humidity both or separately set to 0.
After this, the complete dataset was pre divided in a training- and testing set. Where 20% of the total dataset was pseudo-randomly taken to form a test set. One condition that was applied during this process is that 20% of the data in the test set is anomalous. All the leftover anomalies were left in the training set. This resulted in the division visible in Table 3.4.
The distribution of the two datasets can be found in Figure 3.2. It can be seen that for the anomalous data the mean is closer to 0 for all sensors, which is due to the nature of the anomalous data where

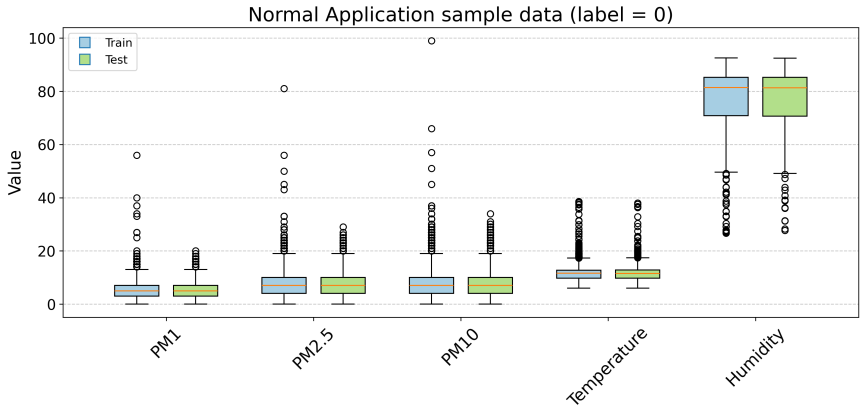
Table 3.3: Overview of anomalous samples in measured dataset.

Anomalous event:	**Time of event:**	**Nr. of Samples**	**Details:**
Simple DOS (label 1)	From: 2025-11-06 22:29:51 To: 2025-11-06 22:33:00	1297	Client was sending correct measurements without a rate limiter. This HTTP flooding attack is inspired by the most common form of attack present in the NSL-KDD dataset related to HTTP traffic: the backlog attack [10]. This attack features large amount of messages coming in a short time period, just like the HTTP flood attack.
Failing Temp/Humidity Sensor (label 2)	From: 2025-11-06 21:37:55 To: 2025-11-07 10:03:22	139	Client was sending correct PM measures, but either humidity or temperature was hardcoded to 0 or a constant not representable to reality.
Failing PM Sensor (label 2)	From 2025-11-07 10:04:31 To: 2025-11-07 17:23:20	89	Client was sending correct Temperature and Humidity measures, but either any PM value was hardcoded to 0 or a constant not representable to reality.
Failing Temp/Humidity and PM Sensor (label 2)	From: 2025-11-07 16:25:02 To: 2025-11-07 20:15:08	35	Temperature, Humidity and PM values were all set to 0.
Total:		1560	Label 1: 1297, Label 2: 263

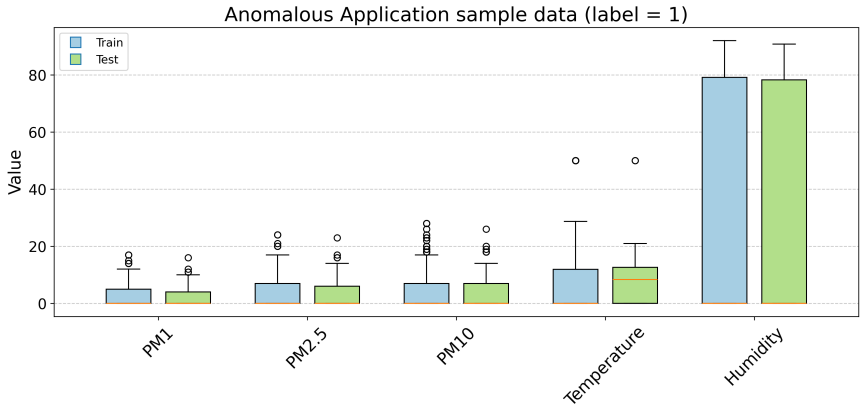
Table 3.4: Amount and type of anomalies in the AM dataset

Type of sample:	**Count Training data:**	**Count Test data:**
Normal data	6573	1642
Broken PM sensor (sends only 0)	497	127
Broken Temperature sensor (sends only 0)	91	19
Broken Humidity sensor (sends only 0)	79	29
Both Temperature and Humidity sensors are broken (both send 0)	375	81
Total:	7423	1855

broken sensors are represented with a 0 in their recording value. More in-depth statistics about the measured AM data can be found in Table 3.5. Besides the WD, these statistics prove to be relevant in the comparison of this dataset to the generated dataset as will be discussed in the next paragraph.



(a) Distribution of normal data.



(b) Distribution of anomalous data.

Figure 3.2: Distribution of normal and anomalous data in the datasets for the AM. The value on the y-axis is the unit of the measurement belonging to the feature type as provided in Section 3.1.1.

Table 3.5: Statistics of recorded dataset from the weather stations. Only normal data (i.e. data with label 0) is considered in these statistics. Data is separated per location of the weather stations. One row is added in the end with some overall statistics.

Location:	**PM1,0 (avg/std):**	**PM2,5 (avg/std):**	**PM10 (avg/std):**	**Temperature (avg/std):**	**Humidity (avg/std):**
Den Hoorn	3.6/3.07	4.8/4.36	4.9/4.85	12.0/2.53	78.0/10.24
Dordrecht	7.3/3.57	10.3/5.18	10.8/6.18	12.0/1.75	77.5/7.55
Lage Zwaluwe	4.5/2.37	6.2/3.44	6.2/3.54	12.2/3.65	76.9/9.34
Wassenaar	5.0/3.43	6.9/4.99	7.1/5.72	12.0/2.44	74.2/8.63
Total:	5.1/3.44	7.1/4.98	7.3/5.62	12.0/2.68	76.6/9.07

As was mentioned in the start of this subsection, the WD is an important metric to compare the later generated data to the real-recorded measurement data using formula 2.13. To set a threshold of what an acceptable distance would be, the WD between two subsets of the AM data could be used. In order to take a single ϵ to predict the maximum ratio of generated data to be used to still train a reliable IDS, the normalized values of the measurements will be used to do this estimation. To find out which would be the maximum threshold to use, the following two calculations will be performed:

- The WD between each combination of two days of one measurement station, so per 24 hours;
- The WD between each combination of two measurement stations over the whole measurement period.

For now, the tolerance will only be calibrated by measuring the WD between independent subsets of the real data. This real–real divergence captures the natural variability present in genuine measurements. This baseline will then be compared to the WD between real and generated samples. As long as the real–synthetic divergence remains on the same order as, or below, the typical real–real divergence, it is to be expected that the induced mixture distribution will remain within an acceptable generalisation regime. This yields the following maximum WDs, visible in Table 3.6:

Table 3.6: Maximum WDs of normalised real-recorded AM data. Only data with label '0' (i.e. non-anomalous recordings) are taken into account.

Application metric:	**Max. WS dist. per 24 hr blocks**	**Block with maximum (start times of the two blocks. Each block is 24 hours)**	**Max. WS dist. between locations**	**Two locations with max distance**
Temperature	0.16	Den Hoorn: 2025-11-04 19:37:09 → 2025-11-07 19:38:02	0.02	Dordrecht <> Lage Zwaluwe
Humidity	0.31	Den Hoorn: 2025-11-01 19:34:23 → 2025-11-04 19:37:09	0.06	Den Hoorn <> Wassenaar
PM1,0	0.22	Dordrecht: 2025-11-04 19:36:51 → 2025-11-07 19:39:46	0.07	Den Hoorn <> Dordrecht
PM2,5	0.23	Dordrecht: 2025-11-04 19:36:51 → 2025-11-07 19:39:46	0.07	Den Hoorn <> Dordrecht
PM10	0.22	Dordrecht: 2025-11-04 19:36:51 → 2025-11-07 19:39:46	0.06	Den Hoorn <> Dordrecht

As can be seen in Table 3.6, the maximum WD is **0.31**. This WD on its own is a good bounding threshold to later on compare the WD between the real and generated datasets with. That is why for now it is decided that:

$$\epsilon_{max} = 0.31 \tag{3.1}$$

NTT data
The NTT data, that is collected, is inspired by the NSL-KDD dataset that has been mentioned earlier. In an earlier stage of this thesis, the NSL-KDD dataset was indeed used to train the network traffic model (NTM). However, this turned out to be a very unreliable model. Its predictions for the label turned out to be generally opposite to the label belonging to the recordings. More on this will be mentioned later in Section 3.3.5. For the training of the NTT data therefore, the data collected through the weather station application is used. This will be analysed in a similar matter as the AM data. The features present in the recorded dataset, are listed in Table 3.7.
Table 3.8 displays the division between normal and anomalous data.

The 28 features of the NTT dataset are not all numerical. Three of them (protocol_type, service and flag) are categorical items. This offers no problem, since the dataloader in the design takes care of transforming these into a numerical and scaled value. For the numerical data, its statistics are given in Table 3.9.

Table 3.7: Features of the NTM training and testing dataset.

Feature	Description
duration	Duration of the connection.
protocol_type	Network protocol used in the connection (e.g. tcp, udp, icmp).
service	Destination network service on the target host (e.g. http, ftp).
flag	Status flag of the connection at the transport layer.
src_bytes	Bytes sent from source to destination in this connection.
dst_bytes	Bytes sent from destination to source in this connection.
land	1 if source and destination IP/port are the same, 0 otherwise.
wrong_fragment	Number of incorrect IP fragments in this connection.
urgent	Number of packets with the urgent flag set in this connection.
count	Number of connections to the same destination host in the last 2 seconds.
srv_count	Number of connections to the same service in the last 2 seconds.
serror_rate	Fraction of connections in `count` with SYN-related errors.
srv_serror_rate	Fraction of connections in `srv_count` with SYN-related errors.
rerror_rate	Fraction of connections in `count` with REJ (error) flags.
srv_rerror_rate	Fraction of connections in `srv_count` with REJ (error) flags.
same_srv_rate	Fraction of connections in `count` to the same service.
diff_srv_rate	Fraction of connections in `count` to different services.
srv_diff_host_rate	Fraction of connections in `srv_count` to different destination hosts.
dst_host_count	Number of connections to the same destination host.
dst_host_srv_count	Number of connections to the same service on the same destination host.
dst_host_same_srv_rate	Fraction of connections in `dst_host_count` to the same service.
dst_host_diff_srv_rate	Fraction of connections in `dst_host_count` to different services.
dst_host_same_src_port_rate	Fraction of connections in `dst_host_srv_count` from the same source port.
dst_host_srv_diff_host_rate	Fraction of connections in `dst_host_srv_count` to different destination hosts.
dst_host_serror_rate	Fraction of connections in `dst_host_count` with SYN-related errors.
dst_host_srv_serror_rate	Fraction of connections in `dst_host_srv_count` with SYN-related errors.
dst_host_rerror_rate	Fraction of connections in `dst_host_count` with REJ errors.
dst_host_srv_rerror_rate	Fraction of connections in `dst_host_srv_count` with REJ errors.

Table 3.8: Amount and type of anomalies in the AM dataset

Type of sample:	**Count Training data:**	**Count Test data:**
Normal data	6387	1596
DOS attack	1036	259
Total:	7423	1855

It can be seen in the table with statistics that often, the data remains close to 0 or is constant at 0. The features ‘wrong_fragment’ and ‘urgent’ are manually kept at 0, since the way Zeek is configured to collect the data currently does not allow for the flags belonging to these features to be recorded. Writing a dedicated parser or using more elaborate package capturing software would be able to provide this data. But for this thesis it is chosen not to. However, they are still included in the dataset. Similar to the AM data, the NTT data is stored along with details of the device that sent the belonging packet.

Table 3.9: Mean and standard deviation (mean/std) of the training and testing dataset for the NTM. The statistics are split up between the training and testing datasets and separated for each label.

Feature	Train 0	Train 1	Test 0	Test 1
duration	0.12/1.63	0.03/0.01	0.07/0.17	0.03/0.01
src_bytes	431.72/6.98	428.93/1.39	431.93/6.31	428.93/1.34
dst_bytes	181.00/0.00	181.00/0.00	181.00/0.00	181.00/0.00
land	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
wrong_fragment	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
urgent	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
count	1.56/1.89	95.93/16.14	1.53/0.52	96.44/14.68
srv_count	1.56/1.88	95.93/16.14	1.53/0.51	96.44/14.68
serror_rate	0.00/0.02	0.00/0.00	0.00/0.01	0.00/0.00
srv_serror_rate	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
rerror_rate	0.00/0.01	0.00/0.00	0.00/0.00	0.00/0.00
srv_rerror_rate	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
same_srv_rate	0.99/0.09	1.00/0.00	1.00/0.07	1.00/0.00
diff_srv_rate	0.01/0.09	0.00/0.00	0.00/0.07	0.00/0.00
srv_diff_host_rate	0.73/0.26	0.01/0.03	0.73/0.25	0.01/0.03
dst_host_count	28.39/10.55	96.46/14.54	28.25/10.53	96.97/13.14
dst_host_srv_count	27.40/10.06	96.38/14.52	27.34/10.26	96.90/13.13
dst_host_same_srv_rate	0.97/0.05	1.00/0.00	0.97/0.05	1.00/0.00
dst_host_diff_srv_rate	0.03/0.05	0.00/0.00	0.03/0.05	0.00/0.00
dst_host_same_src_port_rate	0.00/0.01	0.00/0.00	0.00/0.01	0.00/0.00
dst_host_srv_diff_host_rate	0.23/0.11	0.02/0.05	0.23/0.12	0.02/0.05
dst_host_serror_rate	0.62/0.23	0.00/0.00	0.62/0.23	0.00/0.00
dst_host_srv_serror_rate	0.60/0.24	0.00/0.00	0.59/0.25	0.00/0.00
dst_host_rerror_rate	0.62/0.23	0.00/0.00	0.62/0.23	0.00/0.00
dst_host_srv_rerror_rate	0.60/0.24	0.00/0.00	0.59/0.25	0.00/0.00

For the sake of readability, there will not be an overview of the features per location. Box plots with the distribution of the features are to be seen in Figures 3.3 and 3.4.

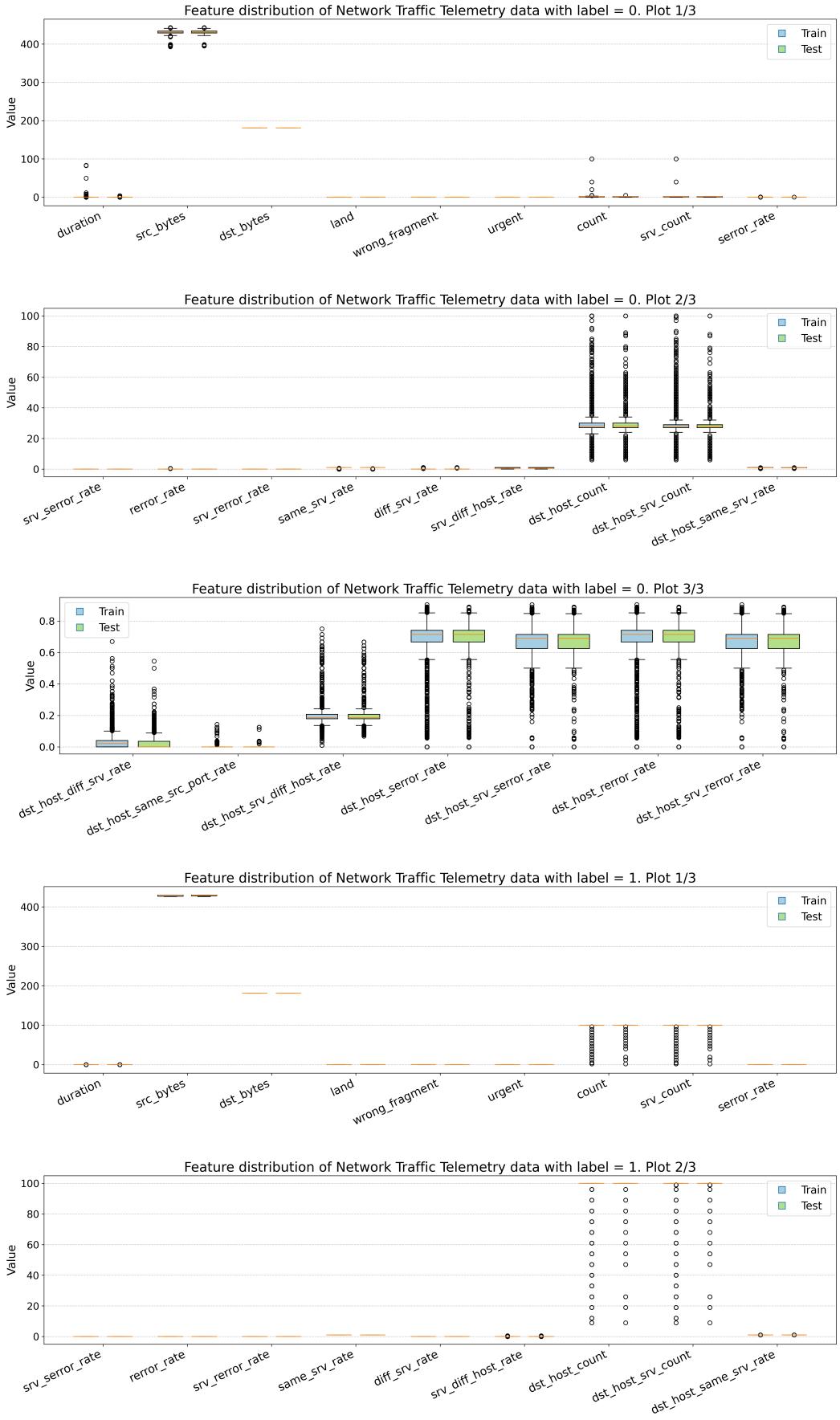


Figure 3.3: Distribution of the training and testing datasets with normal and anomalous data for the NTM. 1/2

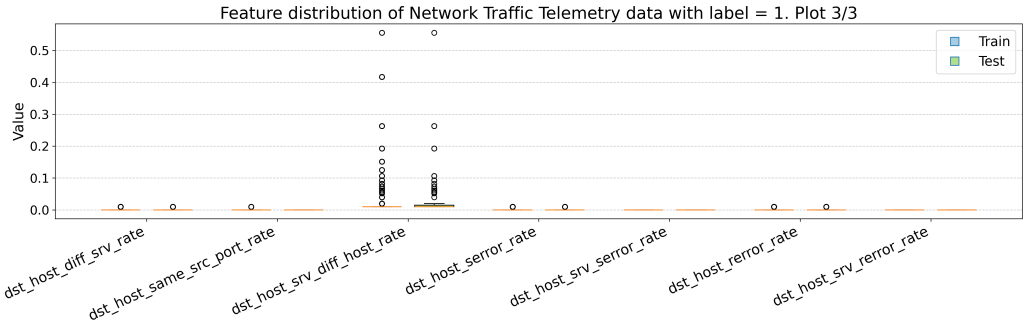


Figure 3.4: Distribution of the training and testing datasets with normal and anomalous data for the NTM. 2/2

With this dataset, the same methods will be used as for the AM data to compute the WDs between subsets of the dataset for normal data. These WDs will then again be used to set a threshold of how much the generated dataset should be able to be mixed with the recorded dataset to still provide a reliable model. The WDs can be found in Table 3.10. For readability reasons, only the features that contain the five largest Wasserstein differences are displayed.

Table 3.10: The top five maximum WDs of normalised real-recorded NTT data. Only data with label '0' (i.e. non-anomalous recordings) are taken into account.

Application metric:	**Max. WS dist. per 24 hr blocks**	**Block with maximum (start times of the two blocks. Each block is 24 hours)**	**Max. WS dist. between locations**	**Two locations with max distance**
dst_host_srv_serror_rate	0.69	Den Hoorn: 2025-11-02 19:35:18 → 2025-11-06 19:34:19	0.04	Den Hoorn <> Wassenaar
dst_host_srv_rerror_rate	0.69	Den Hoorn: 2025-11-02 19:35:18 → 2025-11-06 19:34:19	0.04	Den Hoorn <> Wassenaar
dst_host_serror_rate	0.63	Den Hoorn: 2025-11-03 19:36:15 → 2025-11-06 19:34:19	0.04	Den Hoorn <> Wassenaar
dst_host_rerror_rate	0.63	Den Hoorn: 2025-11-03 19:36:15 → 2025-11-06 19:34:19	0.04	Den Hoorn <> Wassenaar
src_bytes	0.14	Wassenaar: 2025-11-01 19:32:20 → 2025-11-07 19:32:49	0.46	Den Hoorn <> Lage Zwaluwe

It can be seen that the weather station of Den Hoorn is represented in all five largest WDs. This is most likely due to the fact that that weather station was also used to generate the anomalous network traffic data and anomalous AM data. This had as a consequence that as soon as the normal data flow was restarted, the past behaviour of this node is still embedded in its longer range metrics. It can be seen that the top four features are indeed related to longer term effects of a recent HTTP Flood attack. While src_bytes shows a large deviation due to the amount of anomalous data (with small values for the application features) compared to a node that behaved normal.

The largest WD that can be observed in Table 3.10 is **0.69**. This will serve as a threshold to later estimate the minimum ratio of real data that needs to included in the mixed dataset of real and generated samples to still maintain a well performing model. This will be done in the same way as for the AM data by using formula 2.13, where now

$$\epsilon_{max} = 0.69 \tag{3.2}$$

3.3.5. NSL-KDD

As was mentioned earlier in Section 3.3.4, in an earlier stage of the development of this thesis, the NSL-KDD dataset was used as a basis to train the NTM with. However, it turned out after testing this model on the recorded dataset, that this model would not be able to correctly detect anomalies out of the recordings. After more thorough analysis of the two datasets, it came down to most likely the different interpretation of the DOS attacks that were used. In the NSL-KDD dataset, the backlog attack was used [10], where large amounts of data are sent to a server, and the ACK that is supposed to follow after a SYN-ACK is sent by the server is never delivered. Therefore the server's backlog get's flooded with new SYN messages that it at some point will fail to process fast enough. In the recorded dataset, HTTP flooding was performed, where the SYN-ACK is properly replied, but a lot of HTTP messages are sent in a short time. This leads to different characteristics in the features as can be observed in Figure 3.5. In there it can be observed that in case of an attack, the NSL-KDD shows opposite behaviour compared to the recorded data. In the NSL-KDD dataset, a backlog attack is registered as a low amount of packages being received, but with a large payload. Since the SYN-ACK is never completed, it is likely that all SYN requests that follow are considered part of the same message. In the recorded data however, the payload stays the same compared to a normal situation, but the amount of packets being sent rises sharply. Therefore it was decided to move forward with the recorded dataset only instead of the NSL-KDD dataset.

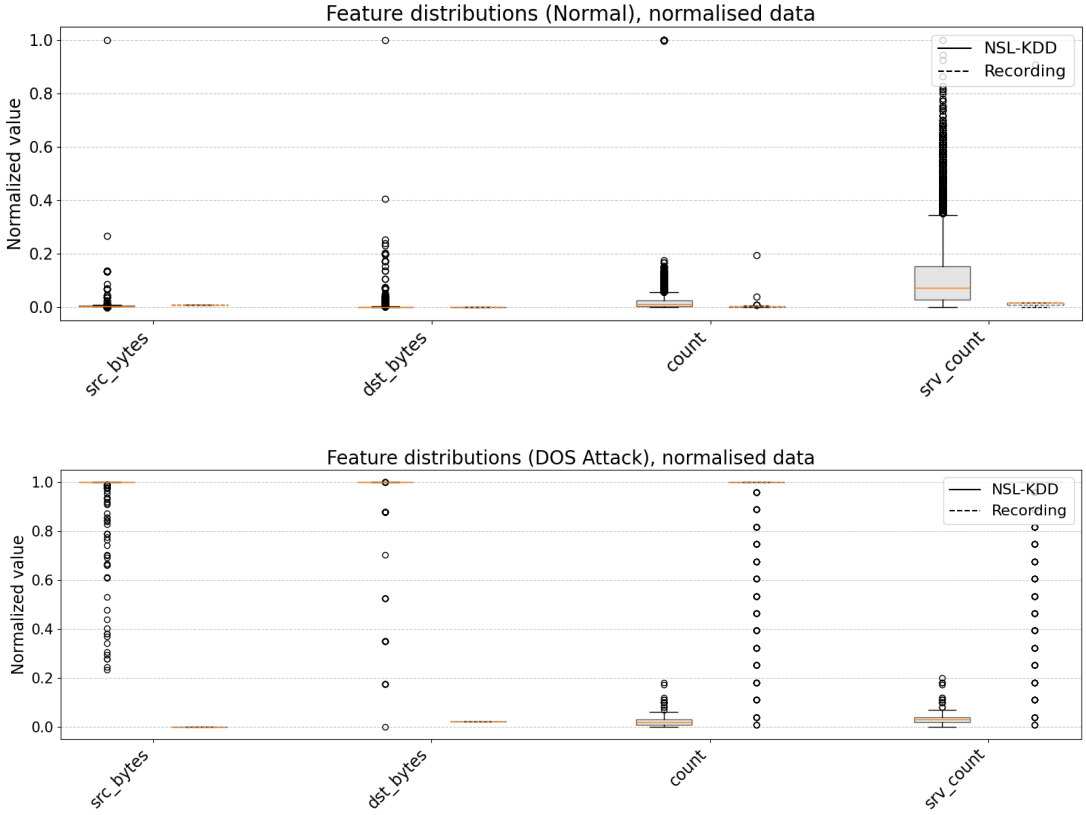


Figure 3.5: Distribution of the features 'src_bytes', 'dst_bytes', 'count' and 'srv_count' compared between the NSL-KDD dataset and the recorded NTT dataset.

3.4. Data Generation

This section will introduce the methods for generating the synthetic data that will be used to mix in with the real-recorded data introduced in Section 3.3.4. As was mentioned in the introduction of this thesis, the LLM will hold the role of generating data for the purpose of this thesis due to its upcoming role for use in the current society as by writing of this thesis. And on top the 'ease of use' does play a great factor in making this decision as well, with the possibility to tune prompts to unimaginable extend as will be discussed in Section 3.4.1. The LLM that will be used in this thesis will be introduced in the next paragraph. The hallucination problem that these models tempt to suffer from, need to be taken into account before training an IDS with generated data. Therefore the data that is generated will be analysed thoroughly in Section 3.4.2

In this thesis, the main LLM-based model that will be used to generate synthetic data will be OpenAI's GPT 5.1 model [21]. This due to its wide popularity as well as many other writings that already have been focused on its behaviour and the use of GPT for several academic researches. GPT is built on the transformer architecture as was described in Section 2.4.2. Its main interaction methods are either through OpenAI's website ChatGPT [21], or through an API that is available for paid subscriptions. OpenAI does make a distinction between users subscribed to a paid plan versus their free to use GPT model. Paying users have access to more prompts in a day and can make use of the latest models that OpenAI has to offer. Due to the amount of testing that needed to be done for the purpose of this thesis, the paid version of ChatGPT was used. To be specific GPT 5.1 is used compared to the free to use, but older model GPT-4o. Which is still very popular at the time of this writing.

3.4.1. Prompts

As was mentioned in Section 2.4.3, prompts should not be created without a proper strategy. Two prompts will be drafted for this thesis for each of the models that will be trained. So two for the AM data and two for the NTT data. The general approach is that each next prompt will be provided with more details than the previous one. After naming the prompts, they will each be compared by the five constraints proposed by [8] and summarised in Section 2.4.3. It should be remarked that in any example data in the prompts that contains IP addresses of client devices, these IP addresses have been manually altered to preserve privacy of the hosts of the weather stations that took part in this experiment. They are not included in the training of the models and are therefore not important to the functionality of the system in the end. The in total four prompts are as follows:

AM data prompts:

For the AM data, two prompts have been written. These are requesting datasets from the LLM, by introducing the measurement set up, how the system performs and how the dataset is structured. The second prompt will, as mentioned, be a bit more restrictive and detailed than prompt 1. Prompt 1 can be read in Figure 3.6 and prompt 2 in Figure 3.7.
These prompts do not seem to be clearly readable from a human perspective. While creating these, it is taken into account that the LLM should be able to receive instructions as clear as possible. Therefore unnecessary language was avoided. In order to analyse these prompts. The first prompt will be held against the principles stated in Section 2.4.3. This is done in Table 3.11.

Prompt 1 Application Data

Create a .csv file filled with samples of a small environment weather station. Connected to the weather station are a humidity sensor, a temperature sensor and a Particular Matter sensor that measures Pm1.0, pm2,5 and pm10 values.
These sensors perform measurements and send their results to a server every 5 minutes. They do this for exactly one week in october. There are 4 different weather stations on 4 different locations. These locations are: Den Hoorn, Dordrecht, Wassenaar and Lage Zwaluwe, all in the Netherlands.
The server collects this data and stores this in a csv in the following format:
timestamp_utc, client_ip, pm1, pm2_5, pm10, temperature, humidity, label, locations
timestamp_utc is a utc timestamp of when the data was recorded and sent to the server
client_ip is a unique ipv4 address that is linked to each weather station
pm1 is the recorded pm1,0 value in micrograms per cubic meter, this is an integer
pm2_5 is the recorded pm2,5 value in micrograms per cubic meter, this is an integer
pm10 is the recorded pm10 value in micrograms per cubic meter, this is an integer
temperature is the recorded temperature in degrees Celsius
humidity is the relative humidity recorded in percentage

label is a label that indicates if data was sent correctly, 0 means everything is fine, 1 means that either one of the sensors is malfunctioning and sending non real recorded data along, this could be 0 or any other number.
locations is the place name in which the device is located. So this would be Dordrecht, Lage Zwaluwe, Wassenaar or Den Hoorn.

Create me a .csv with samples from this system. Use realistic measurements and add a label 0 to this. Also, include some 'anomalous data'. This is data with label 1.

For label 1, insert samples with all of the pm values 0 or any very large number and or the humidity or temperature value 0 or any large number, keep this constant for at least 10 consecutive samples to simulate a stream of broken sensor data.

Make a good mix of real and anomalous data
Make sure that there is only 1 unique IP address per location.
Keep the temperature, humidity and PM values realistic to the time of the year for these locations, do not add temperatures higher than 25 degrees altough it is November. Do not add PM values that are unimaginably large e.g. above 1000
Give me data for the full week, not a subset of the data!

Examples of the samples could be, but are not limited to:
2025-10-31 19:33:29,10.70.20.29,4,5,5,11.552047729492188,82.89918899536133,0,Den Hoorn
2025-10-31
20:46:29,77.175.216.133,5,7,7,11.754798889160156,79.44412231445312,0,Wassenaar
2025-11-07 08:33:21,10.70.20.29,9,13,13,0,0,300,1,Den Hoorn
2025-11-06 21:30:39,10.70.20.29,7,10,10,9.667778015136719,85.28060913085938,0,Den Hoorn

Give me only the .csv file, without any added text.

Figure 3.6: Prompt 1 for the AM data. This prompt leaves a few more items open to interpretation, like the amount of samples.

Prompt 2 Application Data

Create a .csv file filled with samples of a small environment weather station. Connected to the weather station are a humidity sensor, a temperature sensor and a Particular Matter sensor that measures Pm1.0, pm2,5 and pm10 values.
These sensors perform measurements and send their results to a server every 5 minutes. They do this starting at 2025-10-31 19:31:26(UTC) and ending at 2025-11-07 20:00:34 (UTC), round off to the nearest 5 minute time slot if needed. There are 4 different weather stations on 4 different locations. These locations are: Den Hoorn, Dordrecht, Wassenaar and Lage Zwaluwe, all in the Netherlands.
The server collects this data and stores this in a csv in the following format:
timestamp_utc, client_ip, pm1, pm2_5, pm10, temperature, humidity, label, locations
timestamp_utc is a utc timestamp of when the data was recorded and sent to the server
client_ip is a unique ipv4 address that is linked to each weather station
pm1 is the recorded pm1,0 value in micrograms per cubic meter, this is an integer
pm2_5 is the recorded pm2,5 value in micrograms per cubic meter, this is an integer
pm10 is the recorded pm10 value in micrograms per cubic meter, this is an integer
temperature is the recorded temperature in degrees Celsius
humidity is the relative humidity recorded in percentage.
label is a label that indicates if data was sent correctly, 0 means that normal data is sent with a normal rate, 1 means that either one of the sensors is malfunctioning and sending non real recorded data along, this could be 0 or any other number.
locations is the place name in which the device is located. So this would be Dordrecht, Lage Zwaluwe, Wassenaar or Den Hoorn.

Create me a .csv with samples from this system. Use realistic measurements and add a label 0 to this. Also, include some 'anomalous data'. This is data with label 1.
For label 1, insert samples with one or more of the pm values 0 or any large number or the humidity or temperature value 0 or any large number, keep this constant for at least 10 consecutive samples to simulate a stream of broken sensor data. This should roughly make up 50% of the data.
Make sure that at least 50 percent of the data is of the type with label 0.
Make sure that there is only 1 unique IP address per location.
Use realistic measurements to create this dataset. Use online sources to base your creation on. The temperature fluctuates between 4 and 16 degrees Celsius. The humidity between 60 and 92 percent. The PM values all fluctuate between 0 and 25 for realistic data.
Be aware of the natural fluctuation of the data. The temperature drops at night and the humidity decreases when the temperature gets higher.
Give me data for the full time period, not a subset of the data!

Examples of the samples could be, but are not limited to:
2025-10-31 19:33:29,10.70.20.29,4,5,5,11.552047729492188,82.89918899536133,0,Den Hoorn
2025-10-31 20:46:29,77.175.216.133,5,7,7,11.754798889160156,79.44412231445312,0,Wassenaar
2025-11-07 08:33:21,10.70.20.29,9,13,13,0,0,1,Den Hoorn
2025-11-06 21:30:39,10.70.20.29,7,10,10,9.667778015136719,85.28060913085938,0,Den Hoorn
2025-11-06 21:30:47,77.175.216.133,7,10,10,9.661674499511719,85.3144645690918,0,Wassenaar
2025-11-07 03:18:09,86.82.229.87,5,7,7,0,0,1,Lage Zwaluwe

Give me only the .csv file, without any added text.

Figure 3.7: Prompt 2 for the AM data. This prompt is a bit more strict with the requirements it sets to the dataset that the LLM has to provide. It will for example put a minimum on the amount of anomalies, making this an explicit requirement. It will also provide more detail on the measurements that were performed.

Table 3.11: Prompt design rules from Section 2.4.3 applied in Prompt 1 to generate artificial AM data.

Design Rule:	Rule:	Use in prompt 1 (AM data):
1.	**Prompts must explicitly encode the structure of the output.**	1. In the prompt it is stated what columns are to be added to the dataset; 2. It is stated that the output should only be a .csv file, no additional text; 3. The units of all the features are given; 4. Examples of correct output are given.
2.	**The Prompt should provide contextual examples illustrating correct domain behaviour.**	1. Examples are given of correct output. 2. The correct functioning of the measurement system is explained.
3.	**Include explicit constraints describing what the model must do and what it must not do.**	1. Restrictions to the data is given in the form of the exact number of samples that need to be created; 2. The range of features is given (e.g. the maximum temperature); 3. The model is prompted to keep the data realistic to the time of the year and not use random data; 4. The model is prompted not to give just a subset of the data, but the full dataset.
4.	**Specify semantic conditions tied to the domain.**	1. The background to the experiment is given; 2. The different sensors are stated; 3. The time of the year in which the experiment is given to keep the data more realistic; 4. Locations of the measurement stations are given.

NTT data prompts

For the network telemetry data, a similar approach was taken as for the AM data. The two prompts that were created can be read in Figure 3.8 for the first prompt and in Figure 3.9 for the second. The design for the NTM LLM prompt is, just like the prompt for the AM data, held against the design standards proposed in Section 2.4.3. The justification for this can be found in Table 3.12

Prompt 1 Network Traffic Telemetry Data

There is a small weather environment measurement system that has collected several environmental measurements. There are sensors connected to these weather stations.
These sensors perform measurements and send their results to a server every 5 minutes.
They do this for exactly one week in october.
There are 4 different weather stations on 4 different locations.
These locations are: Den Hoorn, Dordrecht, Wassenaar and Lage Zwaluwe, all in the Netherlands.

A server collects this data and records network traffic telemetry data as well using Zeek. It records features that are also present in the NSL-KDD dataset.
Those are:

duration,protocol_type,service,flag,src_bytes,dst_bytes,land,wrong_fragment,urgent,count,
srv_count,serror_rate,srv_serror_rate,rerror_rate,srv_rerror_rate,same_srv_rate,diff_srv_rate,
srv_diff_host_rate,dst_host_count,dst_host_srv_count,dst_host_same_srv_rate,
dst_host_diff_srv_rate,dst_host_same_src_port_rate,dst_host_srv_diff_host_rate,
dst_host_serror_rate,dst_host_srv_serror_rate,dst_host_rerror_rate,dst_host_srv_rerror_rate,label

The only service that my little app uses is http.

Where label indicates if the data was normal or anomalous. Anomalous data is a backlog DOS attack.
Give me a .csv file with records of these measurements where you include both normal entries and anomalous data in the form of the backlog DOS attack.
Give me enough data for a week where every 5 minutes information was sent from 4 different devices.
The data records would for example look like:
0.0472550392150878,tcp,http,SF,431,181,0,0,0,1,1,0.0,0.0,0.0,0.0,1.0,0.0,1.0,27,27,1.0,0.0,0.0,
0.1851851851851851,0.7407407407407407,0.7407407407407407,
0.7407407407407407,0.7407407407407407,0

0.0258569717407226,tcp,http,SF,429,181,0,0,0,2,2,0.0,0.0,0.0,0.0,1.0,0.0,0.5,28,28,1.0,0.0,0.0,
0.1785714285714285,0.7142857142857143,0.7142857142857143,
0.7142857142857143,0.7142857142857143,0

0.0507090091705322,tcp,http,SF,433,181,0,0,0,1,1,0.0,0.0,0.0,0.0,1.0,0.0,1.0,27,27,1.0,0.0,0.0,
0.1851851851851851,0.7037037037037037,0.7037037037037037,
0.7037037037037037,0.7037037037037037,0

1.1995007991790771,tcp,http,SF,439,181,0,0,0,2,2,0.0,0.0,0.0,0.0,1.0,0.0,0.5,27,24,
0.8888888888888888,0.1111111111111111,0.037037037037037,
0.1851851851851851,0.6666666666666666,0.5925925925925926,0.6666666666666666,
0.5925925925925926,0

0.0238661766052246,tcp,http,SF,430,181,0,0,0,100,100,0.0,0.0,0.0,0.0,1.0,0.0,0.01,100,100,1.0,0.0,
0.0,0.01,0.0,0.0,0.0,0.0,1

Give me only the csv. file without any text.

Figure 3.8: Prompt 1 for the NTT data. This prompt leaves a few more items open to interpretation, like the amount of samples.

Prompt 2 Network Traffic Telemetry Data

There is a small weather environment measurement system that has collected several environmental measurements.

There are sensors connected to these weather stations.
These sensors perform measurements and send their results to a server every 5 minutes.
They do this starting at 2025-10-31 19:31:26(UTC) and ending at 2025-11-07 20:00:34 (UTC).
There are 4 different weather stations on 4 different locations.
These locations are: Den Hoorn, Dordrecht, Wassenaar and Lage Zwaluwe, all in the Netherlands.
The server they send their data to is in Den Hoorn.
A server collects this data and records network traffic telemetry data as well using Zeek.
It records features that are also present in the NSL-KDD dataset.

Those are: duration,protocol_type,service,flag,src_bytes,dst_bytes,land,wrong_fragment,urgent,count,
srv_count,serror_rate,srv_serror_rate,rerror_rate,srv_rerror_rate,same_srv_rate,diff_srv_rate,
srv_diff_host_rate,dst_host_count,dst_host_srv_count,dst_host_same_srv_rate,dst_host_diff_srv_rate,
dst_host_same_src_port_rate,dst_host_srv_diff_host_rate,dst_host_serror_rate,
dst_host_srv_serror_rate,dst_host_rerror_rate,dst_host_srv_rerror_rate,label

The only service used in the recordings is http.
Label indicates if the data was normal or anomalous. Anomalous data is a backlog DOS attack. Or any other http attack that could exist.

Give me a .csv file with records of these measurements where you include both normal entries and anomalous data in the form of the backlog DOS attack.
Do not just use random data, but base your created data on general academically accepted sources.
Every 5 minutes, data was sent to the server, when normally behaving
Include all the features, make the data as realistic as possible

The data records would for example look like:
0.0472550392150878,tcp,http,SF,431,181,0,0,0,1,1,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,27,27,1.0,0.0,0.0,
0.1851851851851851,0.7407407407407407,0.7407407407407407,
0.7407407407407407,0.7407407407407407,0

0.0258569717407226,tcp,http,SF,429,181,0,0,0,2,2,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.5,28,28,1.0,0.0,0.0,
0.1785714285714285,0.7142857142857143,0.7142857142857143,0.7142857142857143,
0.7142857142857143,0

0.0507090091705322,tcp,http,SF,433,181,0,0,0,1,1,0.0,0.0,0.0,0.0,0.0,1.0,0.0,1.0,27,27,1.0,0.0,0.0,
0.1851851851851851,0.7037037037037037,0.7037037037037037,0.7037037037037037,
0.7037037037037037,0

1.1995007991790771,tcp,http,SF,439,181,0,0,0,2,2,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.5,27,24,
0.8888888888888888,0.1111111111111111,0.037037037037037,0.1851851851851851,
0.6666666666666666,0.5925925925925926,0.6666666666666666,0.5925925925925926,0
0.0238661766052246,tcp,http,SF,430,181,0,0,0,100,100,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.01,100,100,1.0,
0.0,0.0,0.01,0.0,0.0,0.0,0.0,1

Only 50% of the data should be normal. Make the other 50% anomalous. Keep the anomalous data varied.
In total I want to receive at least enough samples to fit the time period in where every 5 minutes samples were sent with 4 devices and on top of that anomalous data. The generation of anomalous data is allowed to overwrite the requirement for sending normal samples every 5 minutes.
Give me only the csv. file in the form of a download link. Do not write any additional text outside of the .csv file.

Figure 3.9: Prompt 2 for the NTT data. This prompt is a bit more strict with the requirements it sets to the dataset that the LLM has to provide. It will for example put a minimum on the amount of anomalies, making this an explicit requirement. It will also provide more detail on the measurements that were performed.

Table 3.12: Prompt design rules from Section 2.4.3 applied in Prompt 1 to generate artificial NTT data.

Design Rule:	Rule:	Use in prompt 1 (NTT data):
1.	**Prompts must explicitly encode the structure of the output.**	1. In the prompt it is stated what columns are to be added to the dataset; 2. It is stated that the output should only be a .csv file, no additional text; 3. Examples of correct output are given.
2.	**The Prompt should provide contextual examples illustrating correct domain behaviour.**	1. The application has been introduced into the prompt to provide background information; 2. The anomalies that are present in the NSL-KDD dataset have been listed to be included in the data.
3.	**Include explicit constraints describing what the model must do and what it must not do.**	1. The amount of required samples has been listed; 2. The prompt indicates what amount of the data should be anomalous. 3. It is instructed to solely provide the .csv file, no additional text.
4.	**Specify semantic conditions tied to the domain.**	1. Background information of the application was given; 2. Locations of the clients were provided, this could effect the duration of the connection; 3. Information on the software used to capture the NTT data was given; 4. Information of when attacks happen was given.

Prompt 2 deviates again from the first prompt in that it provides more detail on the information that should be given. In this case this means that for example the exact time was given in between data was sent to the server. It was also explicitly instructed that the data should be realistic and new data should be generated. This data should not be generated at random, prompting the model to use its capabilities as a system to search the internet for more background sources than just randomly generating numbers. The model was also instructed that at least 50% of the samples should consist of anomalies.

3.4.2. Data Analysis

Now that synthetic data has been created by using OpenAI's ChatGPT [21] with the prompts from Section 3.4.1, it is important to analyse these datasets on inconsistencies and usability. For both datasets this will be done by: Verifying correct labelling of features, providing general statistics of the dataset and comparing the WDs of the generated dataset per feature with the real-recorded data.

AM data
First it was verified that there are no discrepancies in the feature labelling of the generated dataset compared to the real-recorded dataset. Second the generated AM data will be compared with the real-recorded AM data. It turned out that when the LLM was asked to estimate a good ratio between normal and anomalous data, it only came up with a fairly short list of anomalous samples. Prompt 2 has solved this issue by specifying a minimum requirement to this. As can be seen in Table 3.13 General statistics of the generated data can be found in Tables 3.14 and 3.15

Table 3.13: Amount and type of anomalies in the generated AM datasets

Type of sample:	**Prompt 1:**	**Prompt 2:**
Normal data	7951	4092
Broken PM sensor	48	1000
Broken Temperature sensor	24	1000
Broken Humidity sensor	0	1000
Both Temperature and Humidity sensors are broken	35	1000
Total:	8064	8092

Table 3.14: Statistics of generated dataset using prompt 1. Only normal data (i.e. data with label 0) is considered in these statistics. Data is separated per location of the weather stations. One row is added in the end with some overall statistics.

Location:	**Temperature (avg/std):**	**Humidity (avg/std):**	**PM1,0 (avg/std):**	**PM2,5 (avg/std):**	**PM10 (avg/std):**
Den Hoorn	11.9/2.93	82.1/4.14	7.0/3.18	9.3/3.90	12.3/4.78
Dordrecht	11.0/2.94	80.0/4.08	9.1/3.22	11.5/3.90	14.6/4.81
Lage Zwaluwe	10.5/2.94	78.0/4.10	8.1/3.31	10.3/3.92	13.3/4.84
Wassenaar	13.0/2.92	85.0/4.05	6.1/3.16	8.5/3.87	11.6/4.76
Total:	11.6/3.08	81.3/4.85	7.6/3.41	9.9/4.06	12.9/4.93

Table 3.15: Statistics of generated dataset using prompt 2. Only normal data (i.e. data with label 0) is considered in these statistics. Data is separated per location of the weather stations. One row is added in the end with some overall statistics.

Location:	**Temperature (avg/std):**	**Humidity (avg/std):**	**PM1,0 (avg/std):**	**PM2,5 (avg/std):**	**PM10 (avg/std):**
Den Hoorn	9.7/2.95	82.7/6.71	7.1/2.99	9.6/3.45	12.1/3.91
Dordrecht	10.2/3.04	80.9/7.06	9.1/3.00	11.6/3.43	14.1/3.89
Lage Zwaluwe	10.0/2.99	81.9/6.93	8.0/2.88	10.5/3.35	13.0/3.74
Wassenaar	9.7/3.04	83.8/6.69	6.2/2.83	8.7/3.25	11.2/3.61
Total:	9.9/3.01	82.3/6.92	7.6/3.12	10.1/3.54	12.6/3.94

It seems that in the case of prompt 1 the temperature and humidity features are quite comparable to those of the recorded dataset. The PM values however are with both prompts higher in general than the recorded pm values. Prompt 2 shows a lower average in the temperatures and a higher relative air humidity. In general the generated data seems more constant than the recorded data. Due to the presence of more anomalous data, the dataset given by the LLM after instructing it with prompt 2 will be used in the upcoming experiments of Chapter 4.

In order to check for the usability of the data for training the LSTM Deep learning model as well as would be possible with the recorded data, and to find the maximum ratio in which the two types of datasets (generated and the real dataset) can be mixed before the functionality of the deep learning model should start to deteriorate, the WDs between the distributions are again important here. These are calculated using the data, this time with both labels 0 and 1. These are provided in Table 3.16:

Table 3.16: WDs between the normalised recorded AM data and the generated data using prompt 2. Both normal and anomalous data were taken into account.

Feature	Normal (label=0)	Anomalous (label=1)
PM1,0	0.0401	0.0581
PM2,5	0.0358	0.0527
PM10,0	0.0525	0.0590
Temperature	0.0357	0.0241
Humidity	0.0470	0.0510

As can be observed, the WDs between the generated data and the real-recorded data in Table 3.16 are overall lower than the maximum WDs found between subsets of the real-recorded data from Table 3.6. This already indicates that the AM data is very 'replaceable' with the generated data without causing a shift in the behaviour of a deep learning model trained with any. To verify the ratio of real to generated data that would cause a shift in the behaviour, formula 2.13 of Section 2.5.3 can now be used. When applying the maximum WD found between the generated data of the two prompts in the formula, this yields the following:

$$D\left(P_{\text{synth}_{prompt2}} \parallel P_{\text{real}}\right) = 0.06 \tag{3.3}$$

$$1 - \alpha \leq \frac{\epsilon_{max}}{0.06} \tag{3.4}$$

With $\epsilon_{max} = 0.31$ this yields:

$$1 - \alpha_{min} \leq \frac{0.31}{0.06} = 5\frac{1}{6} \tag{3.5}$$

Since α_{min} cannot be negative, this means that the AM data should be fully interchangeable with the generated data from prompt 1 and not cause a shift in accuracy. However, since the ϵ_{max} was just an estimation based on the maximum Wasserstein Difference amongst subsamples of the real-recorded data, the accuracy can in the end still be affected.

Network Traffic Data

For the network data, also two datasets have been generated, using the prompts from Section 3.4.1. The headers have been checked and it was verified that the LLM did not make a mistake in the list of features. For the generation of anomalies it can be seen in Table 3.17 that prompt 1 has, just like with the AM data, the problem that there are too few anomalies generated. This is solved by prompt 2 by explicitly stating requirements for the amount of anomalies. Statistics about the generated datasets can be observed in Table 3.18.

Table 3.17: Amount and type of anomalies in the generated NTT datasets

Type of sample:	Prompt 1:	Prompt 2:
Normal data	7675	8088
DOS Attack	389	8088
Total:	8064	16176

The datasets have similar statistics for most of the features. A notable exception to this however, are the statistics for the feature 'srv_diff_host_rate', for prompt 2 this is showing the opposite behaviour from the real data in Table 3.9, where the mean increases for anomalous data. In the dataset generated by prompt 2, this seems to have the opposite effect. The real impact of the generated dataset on training the NTM however, can be better estimated by calculating the WDs between the generated dataset and the real-recorded data. For this, only the generated dataset generated by the LLM using prompt 2 will be used. This is due to the fact that the dataset generated by the LLM from prompt 1 has an insufficient amount of anomalies and is therefore considered not usable. The result from calculating these WDs can be observed in Table 3.19.

Table 3.18: Mean and Standard variation of numerical features of the LLM generated datasets for the NTT data compared. Only samples with label 0 are taken into account here.

Feature	Prompt 1, label = 0	Prompt 1, label = 1	Prompt 2, label = 0	Prompt 2, label = 1
duration	0.61/0.34	0.11/0.05	0.23/0.22	0.13/0.10
src_bytes	429.82/17.53	410.12/29.26	448.54/81.13	184.29/191.89
dst_bytes	190.27/17.64	185.03/20.10	178.97/59.35	75.69/77.94
land	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
wrong_fragment	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
urgent	0.00/0.00	0.00/0.00	0.00/0.00	0.00/0.00
count	5.55/2.88	90.92/18.04	15.17/8.33	74.92/14.65
srv_count	5.48/2.87	88.99/17.20	14.91/8.37	74.76/14.70
serror_rate	0.00/0.00	0.00/0.00	0.01/0.01	0.96/0.04
srv_serror_rate	0.00/0.00	0.00/0.00	0.01/0.01	0.96/0.04
rerror_rate	0.00/0.00	0.00/0.00	0.01/0.01	0.91/0.08
srv_rerror_rate	0.00/0.00	0.00/0.00	0.01/0.01	0.91/0.07
same_srv_rate	0.85/0.09	0.97/0.01	0.83/0.09	0.98/0.02
diff_srv_rate	0.15/0.09	0.03/0.01	0.04/0.03	0.01/0.01
srv_diff_host_rate	0.65/0.20	0.05/0.03	0.09/0.06	0.38/0.14
dst_host_count	34.83/14.77	99.89/11.95	108.52/52.10	167.44/50.61
dst_host_srv_count	22.42/11.33	99.53/11.94	80.38/40.49	166.93/50.15
dst_host_same_srv_rate	0.75/0.14	0.97/0.01	0.80/0.11	0.98/0.02
dst_host_diff_srv_rate	0.25/0.14	0.03/0.01	0.07/0.05	0.01/0.01
dst_host_same_src_port_rate	0.15/0.09	0.02/0.01	0.17/0.10	0.96/0.03
dst_host_srv_diff_host_rate	0.70/0.12	0.03/0.01	0.12/0.08	0.05/0.04
dst_host_serror_rate	0.70/0.12	0.00/0.00	0.01/0.02	0.97/0.03
dst_host_srv_serror_rate	0.70/0.12	0.00/0.00	0.01/0.02	0.97/0.03
dst_host_rerror_rate	0.70/0.12	0.00/0.00	0.01/0.02	0.93/0.06
dst_host_srv_rerror_rate	0.70/0.12	0.00/0.00	0.01/0.02	0.93/0.06

The WDs in Table 3.19 were calculated by first normalising the recorded dataset and applying the same scaling to the generated dataset. By normalising the data, the differences between the features themselves become comparable. Applying the same scaling weights to the generated dataset preserves the offset in the data, which could have an impact on the training of the NTM. When comparing the WDs with the previously determined threshold of $\epsilon_{max} = 0.69$ in Section 3.3.4, it can be observed that the WDs in this comparison are overall higher. With a few of them even exceeding 1. For estimating the ratio of real data that is needed to keep the performance of the NTM similar to when it is purely trained with real-recorded data, this yields the following:

$$D\left(P_{\text{synth}_{prompt2}} \parallel P_{\text{real}}\right) = 5.42 \tag{3.6}$$

$$1 - \alpha \leq \frac{\epsilon_{max}}{5.42} \tag{3.7}$$

With $\epsilon_{max} = 0.69$ this yields:

$$1 - \alpha_{min} \leq \frac{0.69}{5.42} = 0.13 \tag{3.8}$$

For α_{min} this yields:

$$\alpha_{min} \geq 1 - 0.13 = 0.87 \tag{3.9}$$

This means that at least 87% of the dataset used to train the NTM should consist of real data. to keep the performance similar. However, this is just an estimation. The WD of 5.42 only appears in the anomalous samples for the feature 'src_bytes'. The importance of this feature is determined by how the LSTM trainer will apply weights to the different features. Therefore it could well be that a similar performance of the model will still be preserved even with lower ratios of real data present in its training set.

Table 3.19: WDs between the normalised recorded NTT data and the generated data using prompt 2. Both normal and anomalous data were taken into account.

Feature	Normal (label=0)	Anomalous (label=1)
duration	0.0024	0.0012
src_bytes	1.2446	5.4164
dst_bytes	0.0000	0.0000
land	0.0000	0.0000
wrong_fragment	0.0000	0.0000
urgent	0.0000	0.0000
count	0.1365	0.2318
srv_count	0.1340	0.2334
serror_rate	0.0092	0.9552
srv_serror_rate	0.0000	0.0000
rerror_rate	0.0162	1.8290
srv_rerror_rate	0.0000	0.0000
same_srv_rate	0.1672	0.0248
diff_srv_rate	0.0455	0.0126
srv_diff_host_rate	0.6345	0.3676
dst_host_count	0.8524	0.7593
dst_host_srv_count	0.5636	0.7537
dst_host_same_srv_rate	0.2472	0.0358
dst_host_diff_srv_rate	0.0533	0.0176
dst_host_same_src_port_rate	1.1978	6.7388
dst_host_srv_diff_host_rate	0.1500	0.0471
dst_host_serror_rate	0.6737	1.0675
dst_host_srv_serror_rate	0.6467	1.0687
dst_host_rerror_rate	0.6715	1.0257
dst_host_srv_rerror_rate	0.6462	1.0275

3.5. Intrusion Detection System

The main goal of this thesis as proposed in Chapter 1 is to test the influence of LLM generated data on training an AM and NTM for an IDS. Therefore an application specific IDS was designed. This IDS will live on top of the Environmental measurement application, where it checks incoming HTTP traffic from the distributed measurement stations and runs the Application and NTT data belonging to incoming packets through its models to assign a certain trust score to each node. This trust score can be used to automatically block nodes from using the server's service, if it turns out a node is showing anomalous behaviour, which is to be shown by a trust score dropping below a certain threshold. First the architecture of the IDS will be introduced in Section 3.5.1, after which the software used to run it is discussed in Section 3.5.2. Last but not least, the voting algorithm that is responsible for producing the final trust score is discussed in Section 3.5.3.

3.5.1. Architecture

The IDS consists of several subsystems. These work closely together in tasks like training models, preparing incoming samples for anomaly prediction by the models and eventually translating the output of the models into action. A small GUI is also made to display the nodes' trust scores and to commit manual interventions if needed. A complete overview of the system can be seen in Figure 3.10.

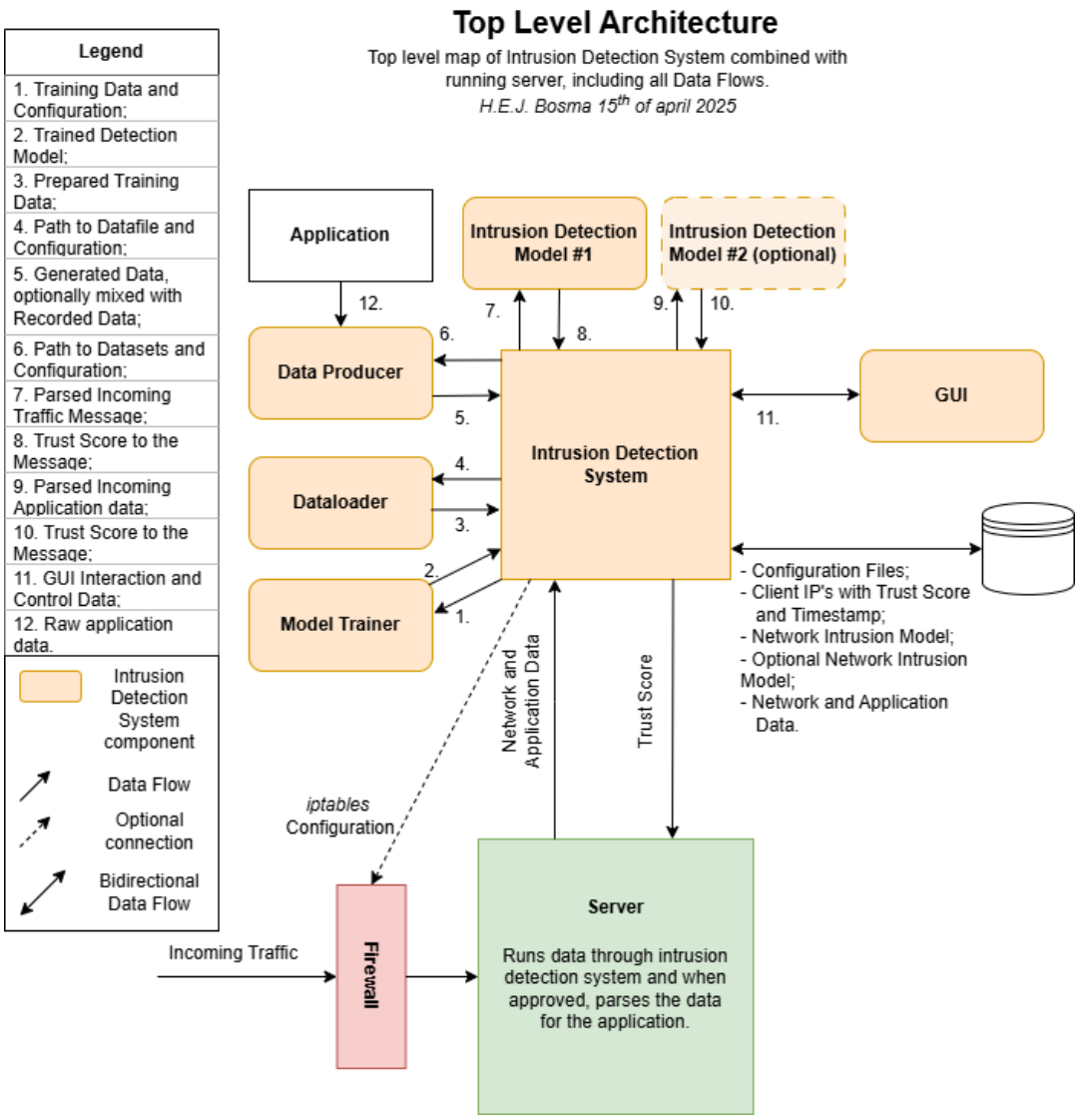


Figure 3.10: Top level architecture of the Intrusion Detection System.

To get introduced briefly to what each component in the IDS is responsible for, here follows an overview:

- **Intrusion Detection System:** This is the heart of the IDS, this controller (that can be partially controlled by the GUI) is responsible for receiving either raw data from the server if configured that way, or receiving prepared data through the data producer from the Application. It is responsible for running the incoming data packets by the trained model(s) and collecting their scores. It is also home to the voting mechanism in Section 3.5.3;

- **Model Trainer:** The model trainer takes datasets from the Dataloader through the main controller and trains a model based on the configuration passed by the main controller as well. It has the capability of reloading a previous trained model and train this model further with new additions on the dataset;

- **Dataloader:** The Dataloader can receive datasets from the main controller by passing paths to their respective csv files. It can mix two datasets and derive test sets by itsself. Separate test sets can also be provided to the Dataloader in order to be scaled accordingly to the training data that was provided earlier. Samples can also be passed one by one to be scaled in order to run them by the trained models;

- **Data Producer:** This is a highly application specific part of the IDS, it is meant to extract raw data from the Application and parse this into a dataset that can be used by the IDS. In the context of the Environment measurement Application this Data Producer for example parses raw Zeek logs into the NTT format required for the NTM;
- **Intrusion Detection Model(s):** The models are responsible for offering predictions on the incoming samples. They will return their predictions in the form of logits, that the main controller will transform into a combined trust score through its voting system;
- **GUI:** The GUI allows for the trust scores of the nodes to be tracked live.

3.5.2. Implementation

All modules have been implemented in the Python programming language. Python was chosen primarily because of its strong ecosystem for machine learning and deep learning, in particular the availability of mature libraries such as PyTorch. These libraries are used in this work to construct data loaders, define model architectures, and perform training and evaluation in a concise and reproducible manner.

For the application layer and graphical user interface, the Flask framework is employed. Flask is a lightweight web framework that is well suited for rapid development and for integrating web-based front-ends with existing Python code. In this project, it is used to generate the front-end from templates while orchestrating the underlying Python scripts that handle data processing, model inference, and interaction with the trained models.

3.5.3. Voting Mechanism

The voting mechanism forms a core feature of the IDS proposed in this thesis. The use of a voting mechanism is proposed in related research by Jiang et al. [16]. However, in that research, the actual way in which the voting processes the outcome of the models to a final result, differs from what has been used in this thesis. Jiang et al. proposed a 'majority win' voting technique, where three models cast a binary vote to determine if a sample is considered normal or anomalous. If the majority of the models agree on the sample to be normal, it will pass as a normal sample. If two out of three or all three models decide that the sample is anomalous, the sample will be marked anomalous.

In this thesis a slightly more elaborate method is created where the resulting 'logits' that the models will produce when a sample is presented on its input are combined into a so called trust score. Since all the datasets that are fed into each model in this thesis have binary labelling (either 0 or 1 for normal or anomalous), each model will produce two logits. These logits indicate how likely it is according to the model that a sample is classified with either a 0 or a 1. Per model, first an individual trust score S will be calculated:

$$S = \sigma(\text{logit}_0 - \text{logit}_1) = \frac{1}{1 - e^{\text{logit}_0 - \text{logit}_1}} \tag{3.10}$$

These trust scores are then combined by taking the average:

$$S_{\text{vote}} = \frac{S_{\text{App}} + S_{\text{Network}}}{2} \tag{3.11}$$

This produces a general trust score where a value close to 1 means the sample is most likely normal and for a value close to 0 it means the sample is most likely anomalous. To make a binary decision, a threshold is used to cast a final vote $vote$:

$$\begin{cases} \text{vote} = 0 & \text{if } S_{\text{vote}} > \text{threshold} \\ \text{vote} = 1 & \text{if } S_{\text{vote}} \leq \text{threshold} \end{cases} \tag{3.12}$$

If a threshold of 0.75 is taken, then the sample will be considered anomalous if one of the two models considers the sample to be anomalous. However, it allows for the possibility that a model rejecting a sample might not be very confident about the decision. This can be observed by the fact that the values of the two logits of that model lie close to each other. In that case, if the other model is very confident about giving the sample a normal label, the sample can still pass as normal.

3.6. Baseline reference models

In order to test the influence of artificially generated data on the performance of the deep learning based IDS, a baseline needs to be established. This baseline will consist of two deep learning models that are trained solely with real reliable data. This means that for the NTM, the recorded NSL-like data will be used. For the AM data model, the recorded AM data will be used. The recorded dataset was therefore split up in two sets. A training and a testing set. The complete test set will be used later to test the performance of the voting system. The division was made randomly, but in such a way that the division of anomalous and normal data was as is visible in Table 3.20.

Table 3.20: The division made in the recorded data to generate a test set for the voting model (VM) and a training set for the base models.

Sample label	Train set #	Relative to total size training set [%]	Test set #	Relative to total size test set [%]
0 = normal	5537	74.59	1383	74.56
1 = DOS attack	1036	13.69	259	13.96
2 = Sensor anomaly	850	11.45	213	11.48

3.6.1. Network Traffic Data

For the NTT data, a baseline model will be trained using the findings of Jiang. et al [16]. The reason for this is that the nature of the data that was used in this paper, the NSL-KDD dataset, is similar to the data set that was recorded for this thesis. Furthermore the researchers also made use of a LSTM architecture. In this paper, the authors describe an LSTM RNN with one LSTM layer, an averaging layer to produce one output and a logistic regression layer to map this output to a probabilistic division between the labels called logits. The maximum logit indicates if the prediction is normal (0) or anomalous (1). The exact parameters used for training this model can be found in Table 3.22. The paper discussed that only 10 epochs were used to train the model. To verify that this would be enough, a small test is performed where different amount of epochs, varying from 1 to 150 in increasing steps of 1 have been used to train the model. The impact on the accuracy and other important output metrics as the precision, recall and F1 score are given in Figure 3.11. To the training set that is provided for training the baseline models, a small adjustment is made. This model will train on the NTT data only, therefore features related to the application will be removed. Since anomalies with label = 2 are related to the sensor data only, and no network anomalies were introduced while a sensor was 'malfunctioning', the labels with value 2 are changed manually to 0 for the training of the NTM. five repetitions were performed per number of epochs with different random divisions of training data batches.

It can be seen in the results that starting from 84 epochs, the output metrics stay fairly constant. Therefore 84 epochs will be used in the baseline model. The paper does however not provide the size of the hidden nodes H in the LSTM layer. Since it is this size that determines in how many nodes the LSTM Model stores its 'memory', it is an important metric. In the tests for the optimal amount of epochs, this is therefore kept at a sufficiently large number: 128. In order to determine the optimum size for this hidden layer, a small comparative test is performed with multiple hidden sizes varying from 2 to 128. It is expected that, in order to capture the influence of all the features, a size of at least 28 (the amount of NTT features) would be needed. The results can be found in Figure 3.12.

It can be seen that starting from 4 hidden nodes the output metrics stay more or less constant, which is smaller than the amount of features that are present in the NTT dataset. This could be explained in the distribution of some of the features, that remain at a constant 0 and therefore do not benefit from a memory effect in the model. A size of 4 will be used in the baseline model.
Training a model with these parameters yielded the test performance metrics in Table 3.21

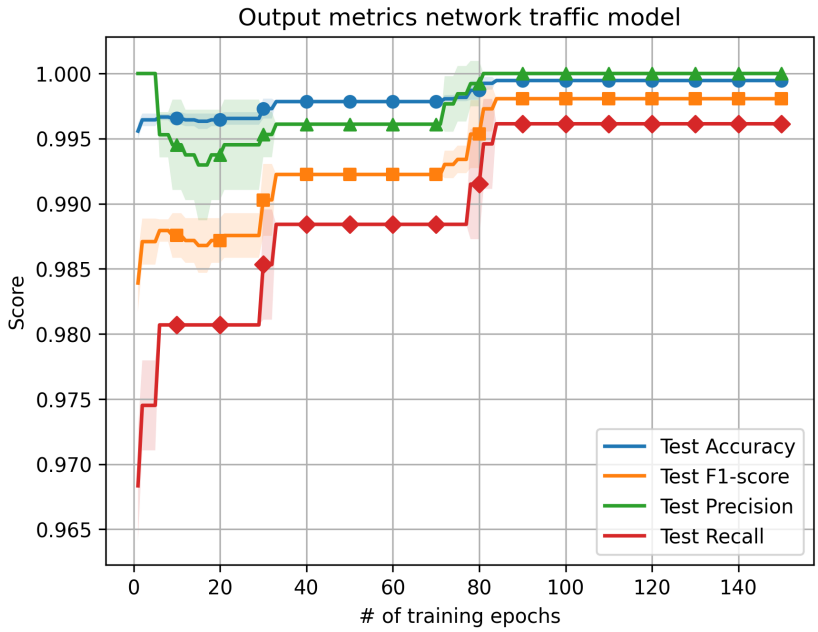


Figure 3.11: The influence of the amount of epochs performed during the training of the baseline model of the NTT data on the output metrics: accuracy, precision, recall and F1-score. The line represents the mean outcome of five repetitions. The shadow around the line indicates the standard deviation of the data. As can be seen in the plots, the output parameters stabilise after 84 epochs, which indicates an optimal trade-off between training costs and performance of the model.

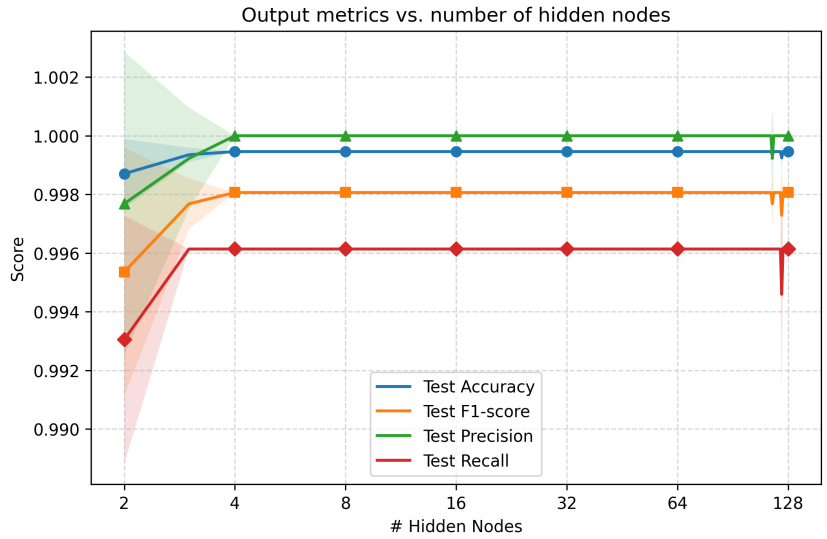


Figure 3.12: The influence of the amount of hidden nodes in the LSTM RNN model for the NTT data on the output metrics after testing this model. These metrics are: accuracy, precision, recall and the F1-score. The output metrics stabilise after using 4 or more hidden nodes. This will be used for the baseline model, since it serves as an optimal trade-off between the complexity of the model and the performance.

Table 3.21: Output performance metrics of the NTM trained with the parameters from Table 3.22

Parameter:	Test Loss:	Accuracy:	Precision:	Recall:	F1-score:
Value:	0.039204	0.988608	0.987850	0.987709	0.987778

Table 3.22: NTT data LSTM model training parameters for the baseline model, based on Jiang et. al [16]. The amount of epochs and the size of the hidden nodes have been determined experimentally.

Parameter type:	**Value:**
Batch size	128
Sequence length	1
Feature size	28
Hidden size	4
Amount of LSTM layers	1
Amount of output classes	2
Learning rate	Starting at 0.1 multiplied with 0.1 per $\frac{1}{3}$ of the epochs. Final Learning rate therefore is 0.001.
Fraction of training data used for testing	20%
Epochs	84

3.6.2. AM data

For the AM data, a similar approach has been taken as for the NTT data in Section 3.6.1. An LSTM RNN has been trained with one LSTM layer, an averaging layer to produce one output and a logistic regression layer to map this output to a probabilistic division between the labels called logits. The maximum logit indicates if the prediction is normal (0) or anomalous (1). For this model, the features that are related to the NTT data have been removed from the training and testing dataset. Since all samples with label 1 (DOS attack) are only relevant to the NTT data based model, they have manually been set to 0 for this model. The exact parameters used for training this model can be found in Table 3.24. The amount of epochs that would converge the model have been determined by training different models with the number of epochs increase from 1 to 150 in steps of 1. five repetitions were performed per number of epochs with different random divisions of training data batches.

The results of this can be seen in Figure 3.13. From the Figure it can be seen that the model stabilises after 145 or more epochs. Therefore 145 epochs will be used to train the model as a trade-off between the computational requirements and the performance of the trained model.
For the amount of hidden nodes, an estimation based on the amount of features in the dataset would predict that five hidden units would be sufficient for this model, since there are five features. When performing trainings with different amounts of hidden nodes and the 145 epochs that were determined earlier, it can be seen in Figure 3.14 that indeed from 8 or more hidden nodes, the output test metrics of the model stabilise. Therefore, 8 hidden nodes will be used.
Training a model with these parameters yielded the test performance metrics in Table 3.23

Table 3.23: Output performance metrics of the AM trained with the parameters from Table 3.24

Parameter:	Test Loss:	Accuracy:	Precision:	Recall:	F1-score:
Value:	0.100894	0.983836	0.995181	0.936508	0.964953

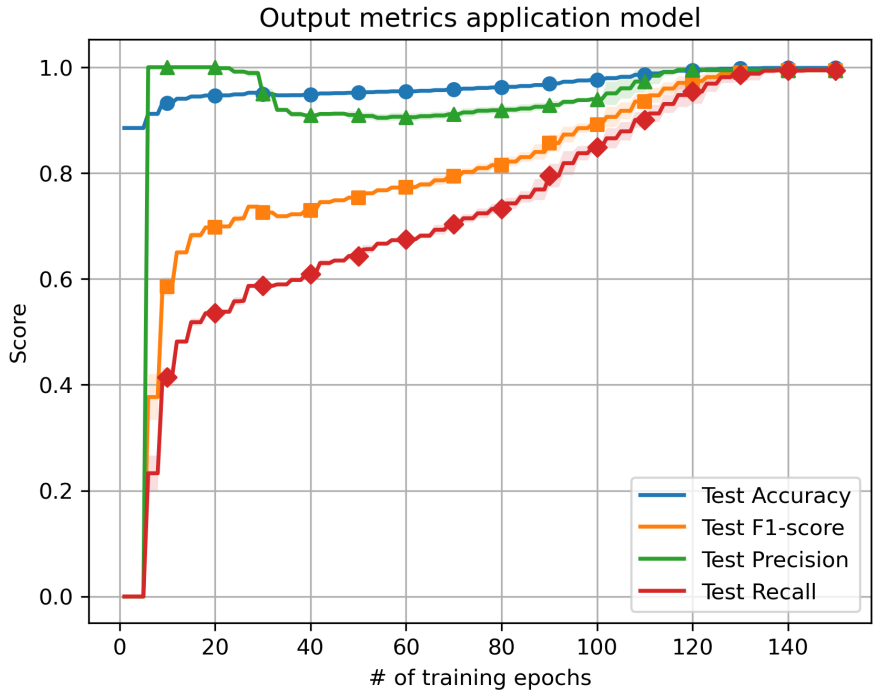


Figure 3.13: The influence of the amount of epochs performed during the training of the baseline model of the AM data on the output metrics: accuracy, precision, recall and F1-score. The line indicates a mean of five repetitions with a shadow around the line indicating the standard deviation. As can be seen in the plots, the output parameters stabilise after 145 epochs, which indicates an optimal trade-off between training costs and performance of the model.

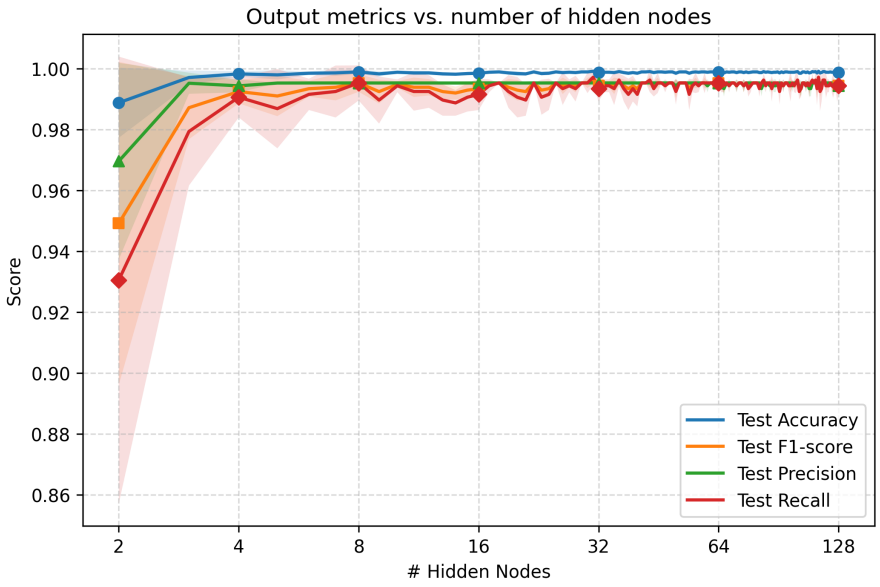


Figure 3.14: The influence of the amount of hidden nodes in the LSTM RNN model for the AM data on the output metrics after testing this model. These metrics are: accuracy, precision, recall and the F1-score. The output metrics stabilise after using 8 or more hidden nodes. This will be used for the baseline model, since it serves as an optimal trade-off between the complexity of the model and the performance.

Table 3.24: AM data LSTM model training parameters for the baseline model. The amount of epochs and the size of the hidden nodes have been determined experimentally.

Parameter type:	**Value:**
Batch size	128
Sequence length	1
Feature size	5
Hidden size	8
Amount of LSTM layers	1
Amount of output classes	2
Learning rate	Starting at 0.1 multiplied with 0.1 per $\frac{1}{3}$ of the epochs. Final Learning rate therefore is 0.001.
Fraction of training data used for testing	20%
Epochs	145

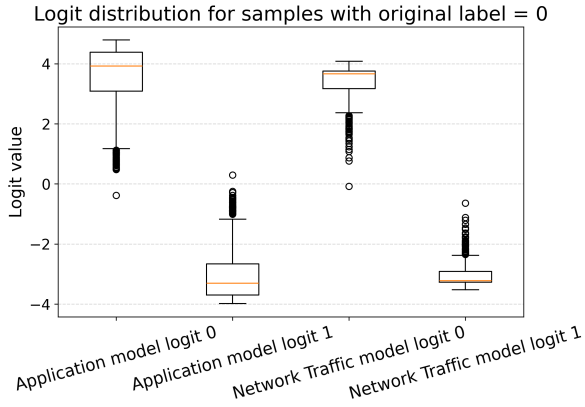
3.6.3. Voting System

Now that the baseline performance of the two models has been evaluated, it is also important to know how they perform combined with each other on the data recordings from the measurement. In order to do this, the two trained models will be put together in in the voting system described in Section 3.5. They will be tested using the test set that was split off from the full recordings when training the baseline models (see Table 3.20). Their voting results will be counted and hold against the label that was connected with the sample to determine the output performance metrics.

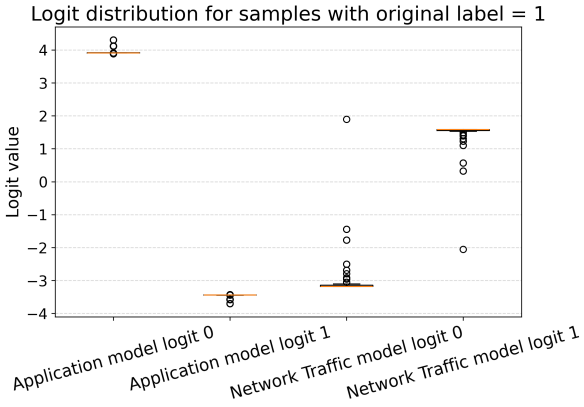
For the final results, the output logits of the two models have been saved. Their distributions can be seen in Figure 3.15. It can be seen that the distributions for the logits for label 0 and 1 are quite far apart for both models, indicating a strong decisiveness of both models. When combining each models' logits in a trust score given by each model (as is done in Figure 3.16a), it can be seen that each model reacts well to the anomalies that are presented for them, to which they specifically have been trained for. So for the AM, it reacts strongly to anomalies with label 2, and for the NTM, it reacts strongly to anomalies with label 1. When combining the trust scores of both models in one overall trust score as is done in Figure 3.16b, then it can be seen that a voters threshold of 0.75 captures the differences between anomalies and normal data quite well with the exception of 3 outliers. This results in the confusion matrix in Table 3.25 and the output parameters in Table 3.26.

Table 3.25: Confusion matrix of the voting system.

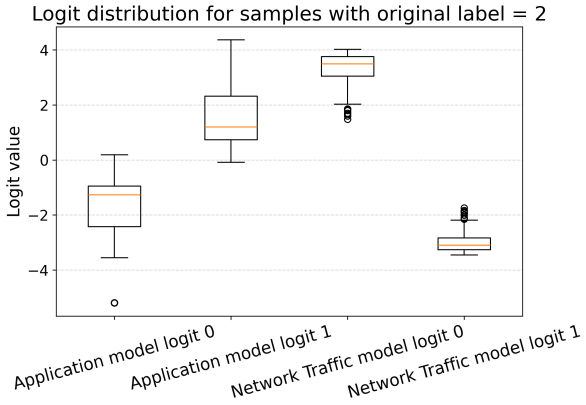
Predicted labels: / **Real labels:**	1 = anomalous	0 = normal
AM:		
1 = anomalous	212	260
0 = normal	1	1382
NTM:		
1 = anomalous	258	214
0 = normal	0	1383
Voting System:		
1 = anomalous	270	2
0 = normal	1	1382



(a) Label of test data = 0



(b) Label of test data = 1

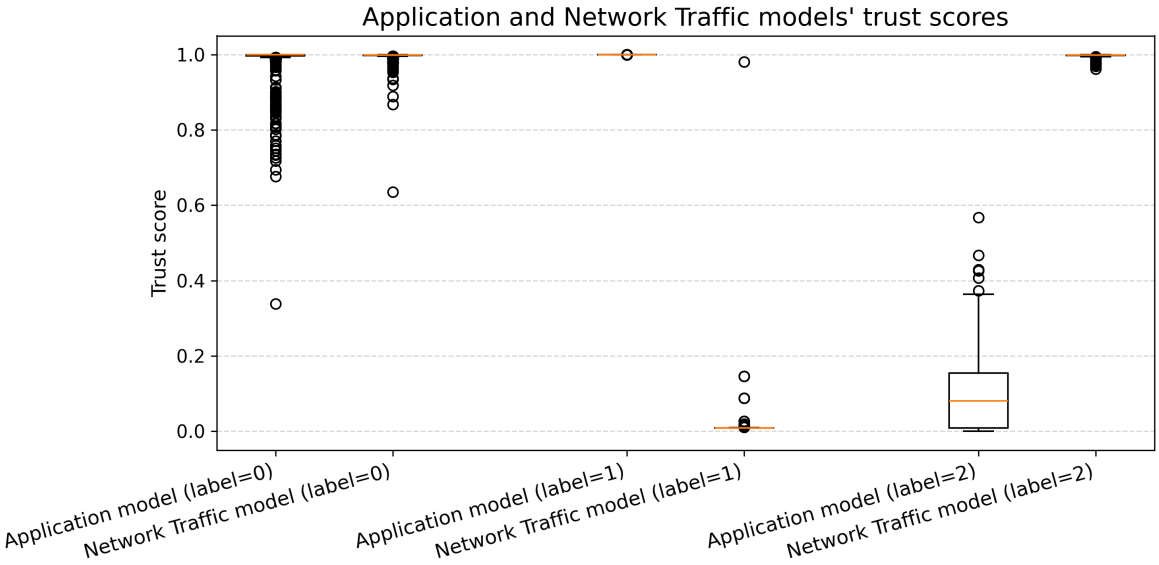


(c) Label of test data = 2

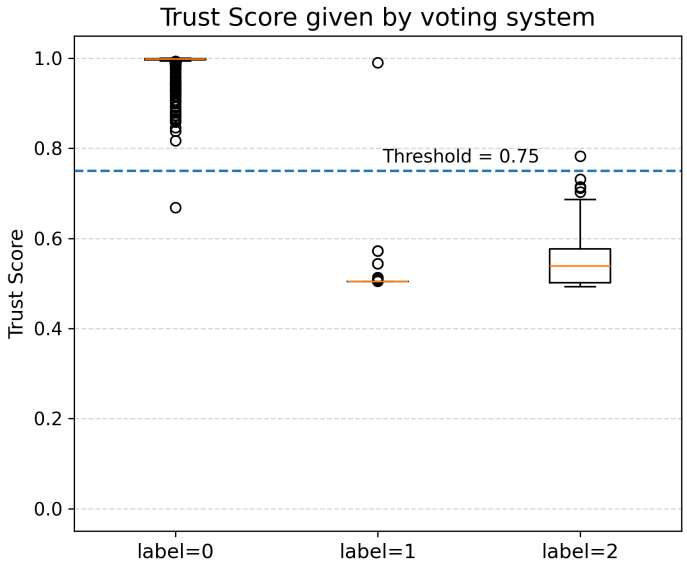
Figure 3.15: Output logits of the AM and the NTM side by side. The output logits indicate a likelihood that the label it represents is actually the label belonging to the sample. In Figure a, both the models give a strong indication towards label 0, which was indeed the label that those samples carried. In Figure b, it can be seen that when the data holds label 1, the NTM indeed reacts to these samples by showing a larger likelihood towards label 1 than label 0. As it should do, since that model is trained in detecting these anomalies. In Figure c, the label is 2. This is a type of anomaly that the AM should pick up. And as can be seen it indeed reacts to these anomalies by showing a stronger likelihood for label 1 (an anomaly is detected).

Table 3.26: Output performance metrics of the VM.

Parameter:	Accuracy:	Precision:	Recall:	F1-score:
AM:	0.859	0.995	0.449	0.619
NTM:	0.885	1.000	0.547	0.707
VM:	0.998	0.998	0.996	0.997



(a) Trust scores of the AM and the NTM. The trust score indicates how trustworthy the source of the samples is when running the latest sample by the models. When data with label 0 is passed, both models indicate that the sample was trustworthy. When data with label 1 is run by the models, the NTM indicates an anomaly by lowering its trust score. When samples with label 2 arrive, the AM shows a lower trust score. It has to be noticed that the variance in its trust score is significantly larger than the variance of the trust scores provided by the NTM.



(b) The overall trust score of the voters system. The average of the trust scores of the two models counts as an overall trust score, where the presence of anomalous data is decided by the trust score falling below a threshold, in this case 0.75. In the case of label 0, one outlier can be seen below the threshold, this will be marked as a false positive. The two outliers above the threshold for label 1 and label 2 are marked as a false negative.

Figure 3.16: Trust scores provided by the two models present in the voting system. Figure (a) shows the raw trust scores of each model, in Figure (b) these are combined into one score.

4

Measurements

In Chapter 1, several problem statements and objectives were proposed in Section 1.3. While some of these objectives have been fulfilled by the data analysis in the previous chapter. The main topic of this thesis is about the influence of LLM generated data on training an AM and a NTM for an IDS. Therefore objective 3b still has to be fulfilled in order to answer research question 3. This will be done by two main experiments (experiment 1 in Section 4.1 and experiment 2 in Section 4.2 of which the results will be shown in Section 4.3. Two experiments will be performed to test the influence of LLM generated data on the baseline models that were described in Section 3.6. The architecture of both the AM and the NTM just as the voting mechanism with it's threshold of 0.75 will therefore be maintained throughout the whole set of experiments.

The two main experiments that will be performed will both test a set of various mixes of real-recorded data and LLM generated data. What differs between the two, is how the data is mixed. In the first set of experiments, real-recorded data in the training dataset will be replaced with increasing amounts of generated data, trying to achieve a complete overview of different ratios of real to generated data, while keeping the total training set size constant. In the second set of experiments, the total size of the training datasets for the Application and NTMs will increase. Data from the LLM generated datasets will be added to increase the ratio of the generated data in the combined training dataset, while preserving the complete information of the recorded dataset. The experiments will be described in more detail in Sections 4.1 and 4.2.

4.1. Experiment 1

In experiment 1, as mentioned, the baseline Application and NTMs from 3.6 will be trained with datasets that will have a constant increase in the ratio of LLM generated data compared to the original recorded data. The original data records are replaced by this generated data, which will keep the total training dataset size constant. The reason for this is that the effect of the generated data can be measured in a more controlled environment. Nothing about the model changes except for the content of the training data.

4.1.1. Set-up

For this experiment both the Application and NTMs will be trained in rounds. In each round, the amount of generated data in the combined training dataset will increase with 1% of the total size of the training dataset. The models' performance will be tested with pre-separated testing datasets for the models. After completing the training, the models will be combined in the voter's model where again a pre-separated dataset is used to test the VM. This will be repeated 5 times where a combined dataset is made with a different subset of the original recorded training data and the generated training data, in the same ratio. After this is completed, the ratio increases with 1%pt and the whole cycle will be repeated until 100% of the training set consists of pure generated data. The parameters of this experiment are collected as well in table 4.1

Table 4.1: Parameters Experiment 1

Parameter	AM	NTM
Training set size (# samples)	7423	7423
Test set size (# samples)	1856	1856
Number of repetitions	5	5
Total number of models trained	500	500
Ratio of generated data to real-recorded data	Start at 0 %to 100% in steps of 1% (either 74 or 75 samples per round)	Start at 0 %to 100% in steps of 1% (either 74 or 75 samples per round)
Input features	5	38
Epochs per training	145	84
Hidden size	8	4

4.1.2. Collected results

For each round of training, some parameters will be calculated and collected. These include the output metrics of the individual model as well as the complete VM, the raw logit output of the models based on their own test sets and the trust scores provided by the voting system. These will be used to produce the following graphs:

1. A plot with the Accuracy, Recall score, Precision score and F1-score over the ratio of generated data over real-recorded data. This will be the average of the scores per round in the experiment. For the Network Traffic and the AM, separate models will also be made with their scores when tested with their own respective test datasets as well as a general plot for the outcome of the VM;
2. A plot with the average and the standard deviation of the raw output logits. These will be separated out over the case where the label of the test data is equal to 0 and when it is equal to 1. They will be taken when the models are separately evaluated on their own test sets, so before the VM. This will be plotted over the ratio of generated data over real-recorded data;
3. A plot with the average and standard deviations of the trust score of the VM over the ratio of generated data over real-recorded data;
4. A plot with the average and standard deviations of the confusion matrices for the Application and the NTMs separately and of the VM in general. Again plotted over the ratio of generated data over real-recorded data.

All this data is stored in 3 rows per run. One for just the AM, one for the NTM and one for the overall VM. This way the output metrics like the accuracy and precision are available of all models and can therefore be analysed separately

4.1.3. Expected outcome

The aim of the experiment is to find out if there is a threshold for the ratio of LLM generated data to real-recorded data from where the performance of the AM, NTM and the Voting system will drop. This performance will be expressed by the outcome metrics of these models. For this thesis it is considered a degradation if one of these metrics drops more than the standard deviation of the same metric in the case where no generated data is mixed in the training set of the model. It is expected that this will happen when the ratio of real data in this training dataset drops below $\alpha_{min} = 0.87$ for the network model and that it will not happen at all for the AM, since the α_{min} for the AM data was smaller than 0.

4.2. Experiment 2

In experiment 2, the baseline Application and NTMs from 3.6 will again be trained with datasets that will have a constant increase in the ratio of LLM generated data compared to the original recorded data. In this experiment however, the LLM generated data samples will be **added** to the recorded data in the combined dataset. Therefore the total size of the training dataset will increase per round. The reason for this is that one of the purposes of this thesis is to explore if LLM generated data could decrease the amount of samples needed for training an IDS model by completing a larger dataset where only part of the data samples comes from 'real' samples.

4.2.1. Set-up

For this experiment both the Application and NTMs will again be trained in rounds. In each round, there will be such an amount of generated data added to the combined dataset, that the total ratio of generated data in the combined dataset has increased with 1%pt. The models' performance will be tested with pre-separated testing datasets for the models. After completing the training, the models will be combined in the voter's model where again a pre-separated dataset is used to test the VM. This will be repeated 5 times where a combined dataset is made with the full recorded dataset and a different subset of the generated training data, in the same ratio. After this is completed, the ratio increases with 1%pt and the whole cycle will be repeated until one of the generated datasets is exhausted. After this, the cycles will continue where only of the other dataset extra samples will keep on being added until that one is exhausted as well. For the first model, the dataset will stay constant at the maximal ratio of generated data over real-recorded data that can be achieved. The parameters of this experiment are to be found in table 4.2. The same parameters are collected as for the first experiment.

Table 4.2: Parameters Experiment 2

Parameter	AM	NTM
Training set size (# samples)	Varying from 7423 until 15,515	7423 until 23,599
Test set size (# samples)	1856	1856
Number of repetitions	5	5
Total number of models trained	270	345
Ratio of generated data to real-recorded data	Start at 0 %to 53% in steps of 1%	Start at 0 %to 69% in steps of 1%
Input features	5	38
Epochs per training	145	84
Hidden size	8	4

4.2.2. Expected outcome

For this experiment, a similar outcome to experiment 1 is expected. However, since the recorded dataset is not being replaced this time, its original structure keeps on contributing to the combined dataset. Therefore the model might become more 'flexible' as soon as more generated data is mixed in, allowing for more samples to be classified as normal. This is because the generated datasets have slightly different statistics compared to the real-recorded dataset, while still being classified as normal data. So instead of a shift, like in experiment 1, an expansion can be expected here. This might cause more false negatives in both models, but especially the AM with its low variation on its features might suffer most from this. The same requirement is hold where lowered performance metrics are considered a degradation if they drop lower than the standard deviation of the metrics in the case no generated data is being used for training yet.

4.3. Experimental Results

After running both experiments and collecting the results, several plots have been made. They will be presented in Section 4.3.1 for experiment 1 and in Section 4.3.2 for experiment 2.

4.3.1. Experiment 1

For experiment 1, the results from the list in Section 4.1.2 have been collected and will be presented here. At first, the behaviour of the raw logit output of the Network Traffic and the AMs are displayed in Figures 4.1 for the NTM and in Figure 4.2 for the AM. It can be seen that for the NTM the logits stay fairly constant up to 48%. From then on the logits 1 and 0 from when anomalous data was presented (true label 1) increase slightly. A sharp change in the logit values can be observed after more than 90% of the training dataset consists of generated data. The '0' logit for true label 0 sharply decreases while the '0' logit for anomalous samples increases sharply. The same applies for the '1' logits. The percentage of generated data in the combined dataset from which the mean reaches outside of the standard deviation range at 0% (the baseline experiment), can be observed in Table 4.3.

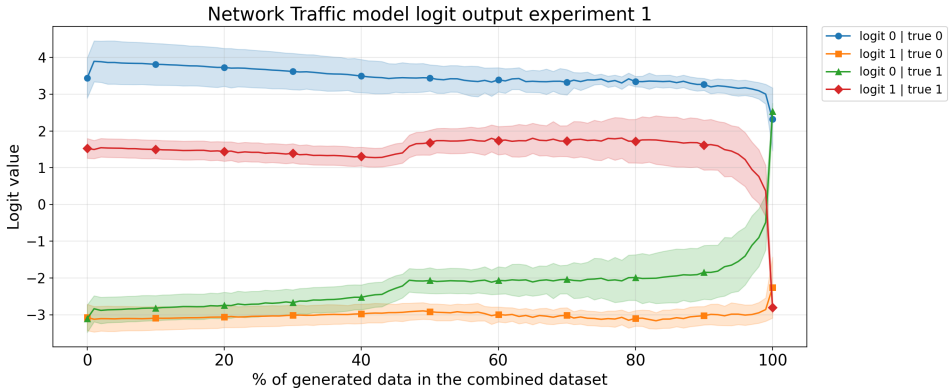


Figure 4.1: The raw logit output of the NTM in experiment 1. The mean of 5 repetitions is plotted for each logit with a shadow around the mean indicating the pooled combination of the standard variation in logits of each run. The logit number indicates for which label the prediction holds, while the number after 'true' indicates the actual label of the samples.

Table 4.3: Percentages to which the mean of the logits of the NTM reach outside of the range of the standard deviation of the baseline model logits.

NTM Logit	Percentage	Average logit value at the percentage	Type of change
Logit 0 for label 0	100%	2.3145	Decrease
Logit 1 for label 0	100%	-2.2596	Increase
Logit 0 for label 1	22%	-2.7079	Increase
Logit 1 for label 1	96%	1.2222	Decrease

Table 4.4: Percentages to which the mean of the logits of the AM reach outside of the range of the standard deviation of the baseline model logits.

AM Logit	Percentage	Average logit value at the percentage	Type of change
Logit 0 for label 0	100%	2.7586	Decrease
Logit 1 for label 0	100%	-2.3022	Increase
Logit 0 for label 1	-	-	Remains within bounds
Logit 1 for label 1	-	-	Remains within bounds

For the AM, the logits remain very constant. Only at 100% the logits that belong to normal samples

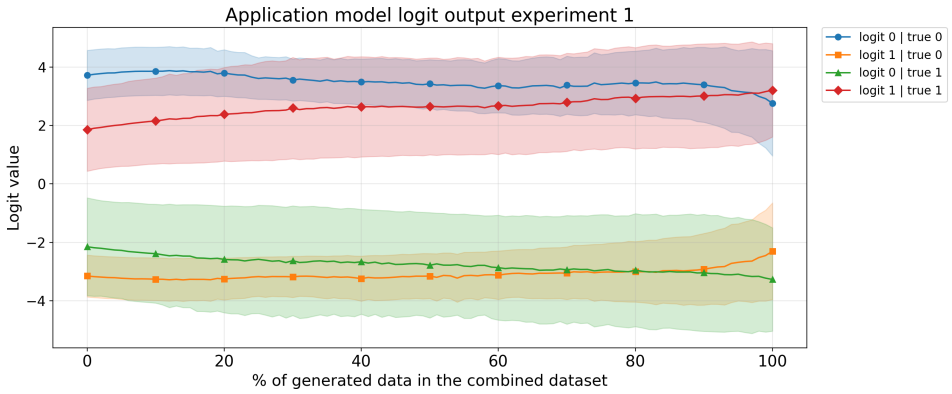


Figure 4.2: The raw logit output of the AM in experiment 1. The mean of 5 repetitions is plotted for each logit with a shadow around the mean indicating the pooled combination of the standard variation in logits of each run. The logit number indicates for which label the prediction holds, while the number after 'true' indicates the actual label of the samples.

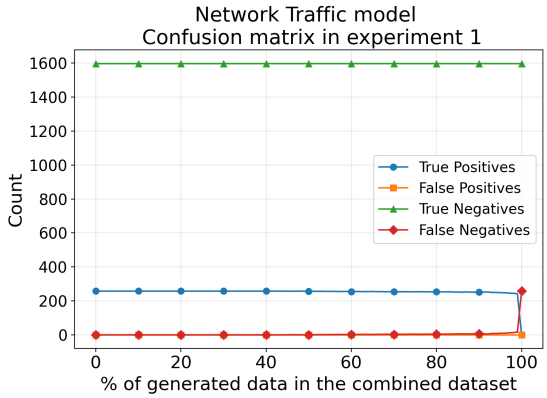
start to reach outside of the standard deviation range at the baseline. The actual average logit values they change to can be observed in Table 4.4.

The next plots to be presented are the confusion matrices of the different models. The confusion matrices have been plotted for all three models separately. The NTM's confusion parameters are displayed in Figure 4.3a, for the AM in Figure 4.3b and the VM in Figure 4.3c. They stay fairly constant with low percentages of generated data mixed in the combined dataset, but they do change. In Table 4.5 are again the percentages on when the data starts to deviate outside of the standard deviation at the baseline.

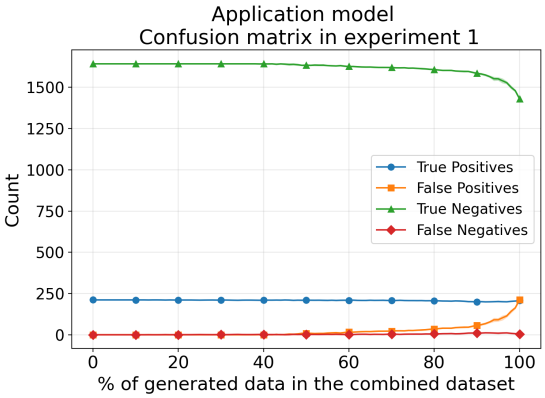
Table 4.5: Percentages at which the mean of the confusion parameters of each model reaches outside of the range defined by the standard deviation of the baseline model parameters, including the corresponding average counts and direction of change.

Model	**Confusion metric**	**Percentage**	**Average count at that percentage**	**Type of change**
Network Traffic	True positives	45%	257.6	Decrease
	False positives	100%	1.8	Increase
	True negatives	100%	1594.2	Decrease
	False negatives	45%	1.4	Increase
Application	True positives	10%	211.8	Decrease
	False positives	39%	1.6	Increase
	True negatives	39%	1640.4	Decrease
	False negatives	10%	1.2	Increase
Voting	True positives	10%	469.8	Decrease
	False positives	39%	1.6	Increase
	True negatives	39%	1381.4	Decrease
	False negatives	10%	2.2	Increase

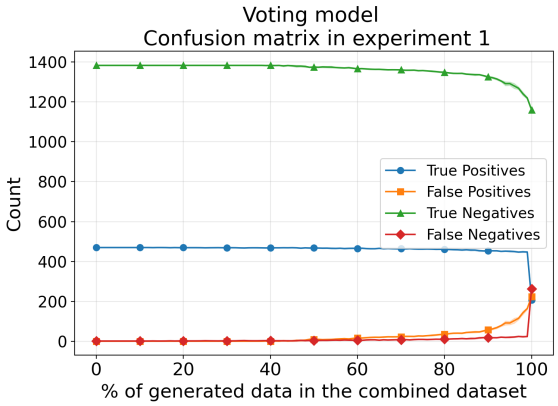
The next data that will be represented are the output metrics of all three models. These include the Accuracy, the Recall score, the Precision score and F1-score of the trained models. The output metrics for the Network Traffic- and the AM, are based on their own test sets. The output metrics of the VM are based on the final test set. These output metrics are plotted in Figure 4.4. Their deviations are shown in Table 4.6. It can be seen that for the NTM, the values stay within range until more than 45% of the combined dataset consists of generated content. However, for the AM and therefore also the VM, the output metrics already start to deviate when 10% or more of the combined training dataset consists of LLM generated data.



(a) The confusion matrix over different percentages of generated data in the combined training dataset of the NTM in experiment 1. The mean of 5 repetitions is plotted for each type with a shadow around the mean indicating the standard deviation over the 5 repetitions.

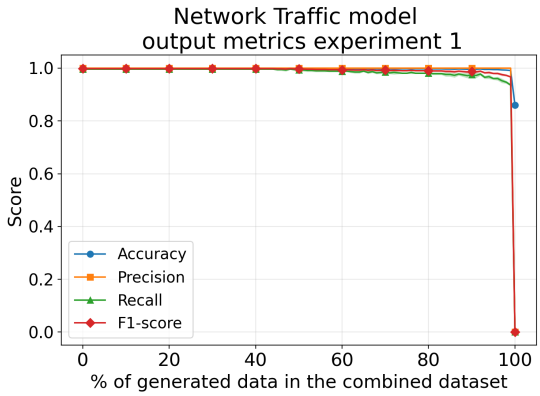


(b) The confusion matrix over different percentages of generated data in the combined training dataset of the AM in experiment 1. The mean of 5 repetitions is plotted for each type with a shadow around the mean indicating the standard deviation over the 5 repetitions.

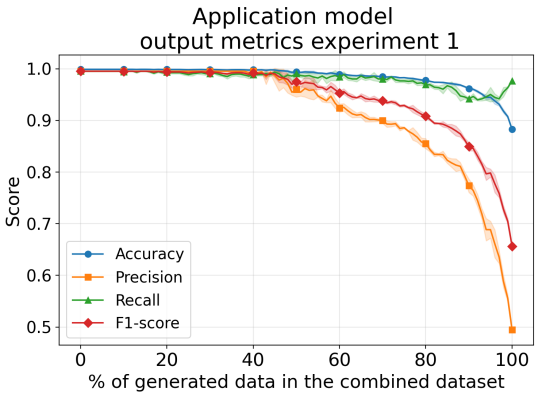


(c) The confusion matrix over different percentages of generated data in the combined training dataset of the VM in experiment 1. The mean of 5 repetitions is plotted for each type with a shadow around the mean indicating the standard deviation over the 5 repetitions.

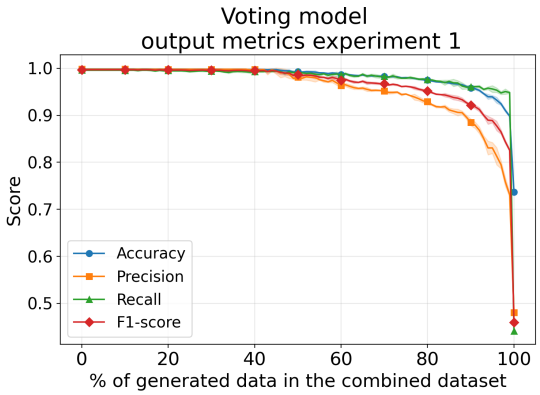
Figure 4.3: Confusion matrices over different percentages of generated data in the combined training dataset for all three models in experiment 1.



(a) The output metrics over different percentages of generated data in the combined training dataset of the NTM in experiment 1. The mean of 5 repetitions is plotted for each metric with a shadow around the mean indicating the standard deviation over the 5 repetitions.



(b) The output metrics over different percentages of generated data in the combined training dataset of the AM in experiment 1. The mean of 5 repetitions is plotted for each metric with a shadow around the mean indicating the standard deviation over the 5 repetitions.



(c) The output metrics over different percentages of generated data in the combined training dataset of the VM in experiment 1. The mean of 5 repetitions is plotted for each metric with a shadow around the mean indicating the standard deviation over the 5 repetitions.

Figure 4.4: Output metrics over different percentages of generated data in the combined training dataset for all three models in experiment 1.

The final plot will contain the overall trust score of the voting system. This will be displayed in Figure 4.5. The percentages from which the trust scores start to deviate outside the range of the standard deviation of the baseline experiment, are displayed in Table 4.7. It can be seen that the trust scores for testing data with labels 0 and 1 eventually move out of the range. This happens for the data with label 0 after 17% of LLM generated data has been mixed in with the real-recorded data for the training datasets. For data with label 1 this happens almost immediately at 1%. The trust score for data with label 2 does stay within boundaries and even decreases slightly when more generated data is mixed in.

Table 4.6: Percentages at which the mean of the output metrics of each model reaches outside of the range defined by the standard deviation of the baseline model parameters, including the corresponding average values and direction of change. Experiment 1

Model	Output parameter	Percentage	Average value at that percentage	Type of change
Network Traffic	Accuracy	45%	0.9992	Decrease
	Precision	100%	0.0000	Decrease
	Recall	45%	0.9946	Decrease
	F1-score	45%	0.9973	Decrease
Application	Accuracy	10%	0.9988	Decrease
	Precision	10%	0.9953	Decrease
	Recall	10%	0.9944	Decrease
	F1-score	10%	0.9948	Decrease
Voting	Accuracy	10%	0.9983	Decrease
	Precision	10%	0.9979	Decrease
	Recall	10%	0.9953	Decrease
	F1-score	10%	0.9966	Decrease

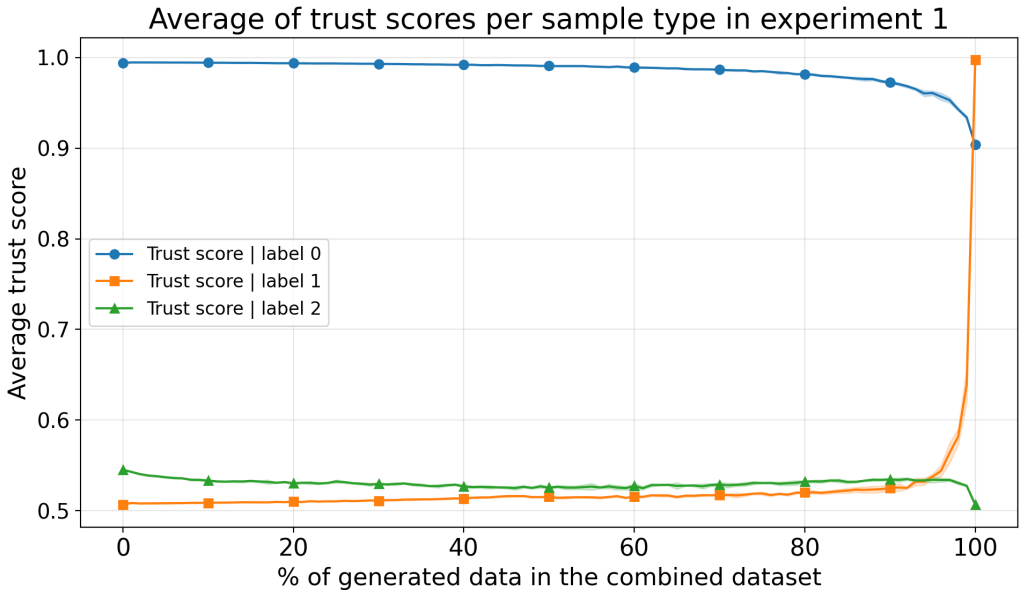


Figure 4.5: Trust score of Voting system compared to the percentage of LLM generated data in the training dataset.

Table 4.7: Percentages at which the trust score of the Voting system starts to reach outside of the range defined by the standard deviation of the baseline model trust scores.

Trust score	Percentage	Average trust score at that percentage	Type of change
Test samples with label 0	17%	0.9938	Decrease
Test samples with label 1	1%	0.5068	Increase
Test samples with label 2	-	-	Always remains lower than baseline experiment

4.3.2. Experiment 2

For experiment 2, the results will be given in a similar matter to the results of experiment one in Section 4.3.1. The difference however, is that in this experiment LLM Generated data has been added on top of the real-recorded data. Therefore, the total percentage of generated data in the graphs will never reach the full 100%, but stay below this.

At first, the behaviour of the raw logit output of the Network Traffic and the AMs are displayed in Figures 4.6 for the NTM and in Figure 4.7 for the AM. It can be seen that for the NTM the logits stay fairly constant up to 44%. From then on the logit 1 from when anomalous data was presented (true label 1) increase slightly, while logit '0' seem to decrease again slightly. The percentage of generated data in the combined dataset from which the mean reaches outside of the standard deviation range at 0% (the baseline experiment), can be observed in Table 4.8.

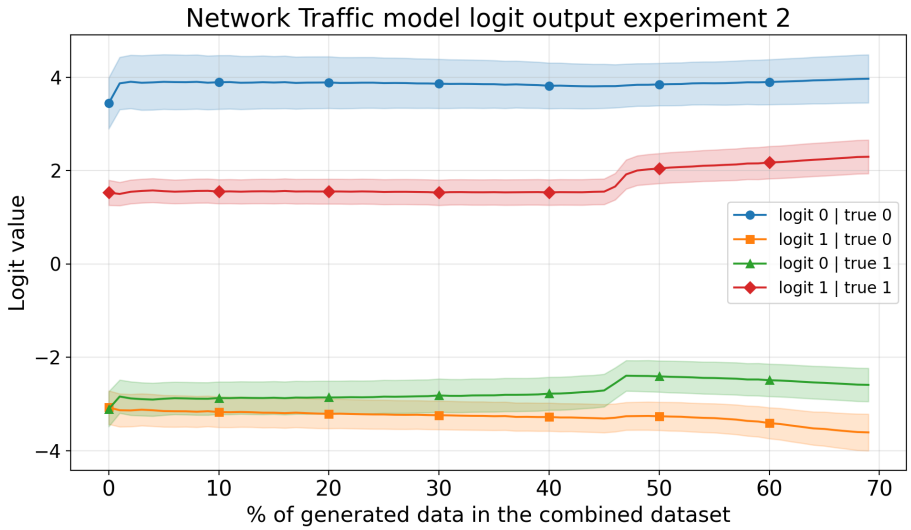


Figure 4.6: The raw logit output of the NTM in experiment 2. The mean of 5 repetitions is plotted for each logit with a shadow around the mean indicating the pooled combination of the standard variation in logits of each run. The logit number indicates for which label the prediction holds, while the number after 'true' indicates the actual label of the samples.

Table 4.8: Percentages to which the mean of the logits of the NTM reach outside of the range of the standard deviation of the baseline model logits.

Logit	Percentage	Average logit value at the percentage	Type of change
Logit 0 for label 0	-	-	Does not reach outside of the range
Logit 1 for label 0	62%	-3.4602	Decrease
Logit 0 for label 1	44%	-2.7328	Increase
Logit 1 for label 1	47%	1.9196	Increase

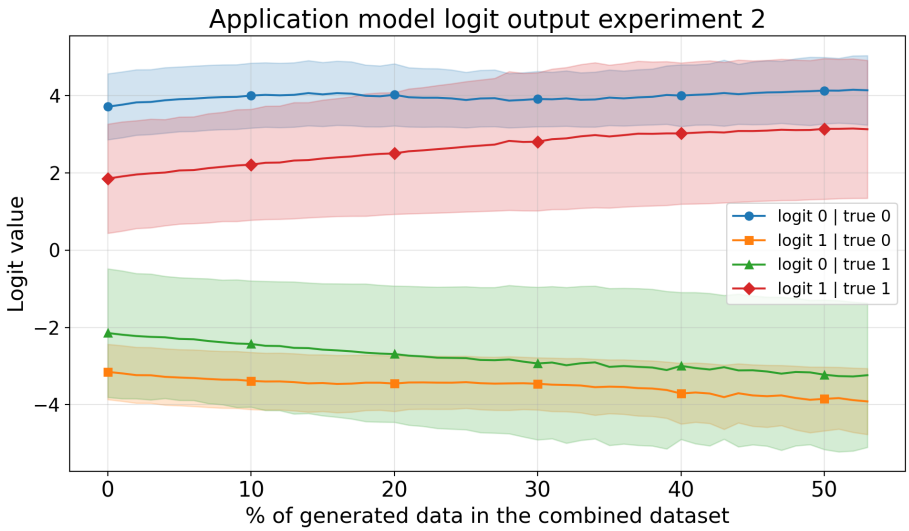


Figure 4.7: The raw logit output of the AM in experiment 2. The mean of 5 repetitions is plotted for each logit with a shadow around the mean indicating the pooled combination of the standard variation in logits of each run. The logit number indicates for which label the prediction holds, while the number after 'true' indicates the actual label of the samples.

Table 4.9: Percentages to which the mean of the logits of the AM reach outside of the range of the standard deviation of the baseline model logits.

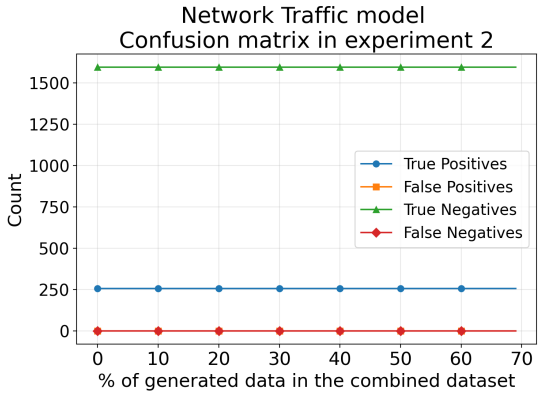
Logit	**Percentage**	**Average logit value at the percentage**	**Type of change**
Logit 0 for label 0	-	-	Remains within bounds
Logit 1 for label 0	49%	-3.8747	Decrease
Logit 0 for label 1	-	-	Remains within bounds
Logit 1 for label 1	-	-	Remains within bounds

For the AM, the logits do seem to shift. However, due to their large standard variations, only logit 1 for label 0 falls out of bounds. It can be observed that the distance between the logits (so the difference between logit 0 for label 0, logit 1 for label 1 and the logits 0 for label 1 and 1 for label 0) increases. Creating a stronger separation between the logits. The actual average logit value to which logit 1 for label 0 changes to can be observed in Table 4.9.

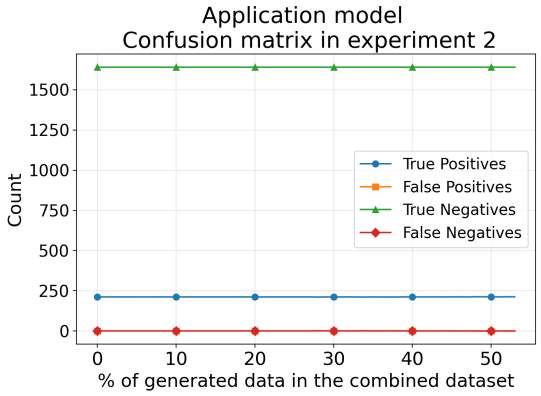
The next plots to be presented are the confusion matrices of the different models. The confusion matrices have been plotted for all three models separately, just like with the results of experiment 1. The NTM's confusion parameters are displayed in Figure 4.8a, for the AM in Figure 4.8b and the VM in Figure 4.8c. They stay fairly constant over the whole range of percentages of generated data mixed in the combined dataset, but the confusion parameters of the AM does change. Table 4.10 shows again the percentages on when the data starts to deviate outside of the standard deviation of the baseline values. It can be seen that when more than 6% of the training data consists of LLM generated data, the True positives slightly increase and the False Negatives slightly decrease.

Table 4.10: Percentages at which the mean of the confusion parameters of each model reaches outside of the range defined by the standard deviation of the baseline model parameters, including the corresponding average counts and direction of change.

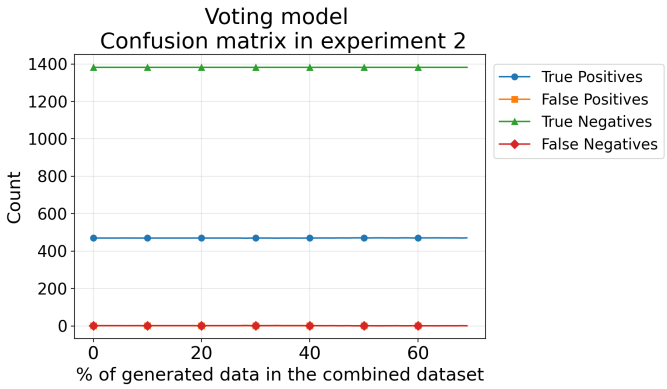
Model	Confusion metric	Percentage	Average count at that percentage	Type of change
Network Traffic	True positives	-	-	Remains within bounds
	False positives	-	-	Remains within bounds
	True negatives	-	-	Remains within bounds
	False negatives	-	-	Remains within bounds
Application	True positives	6%	212.2	Increase
	False positives	-	-	Remains within bounds
	True negatives	-	-	Remains within bounds
	False negatives	6%	0.8	Decrease
Voting	True positives	-	-	Remains within bounds
	False positives	-	-	Remains within bounds
	True negatives	-	-	Remains within bounds
	False negatives	-	-	Remains within bounds



(a) The confusion matrix over different percentages of generated data in the combined training dataset of the NTM in experiment 2. The mean of 5 repetitions is plotted for each type with a shadow around the mean indicating the standard deviation over the 5 repetitions.



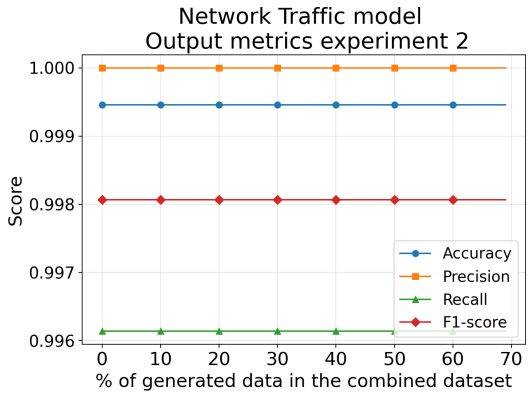
(b) The confusion matrix over different percentages of generated data in the combined training dataset of the AM in experiment 2. The mean of 5 repetitions is plotted for each type with a shadow around the mean indicating the standard deviation over the 5 repetitions.



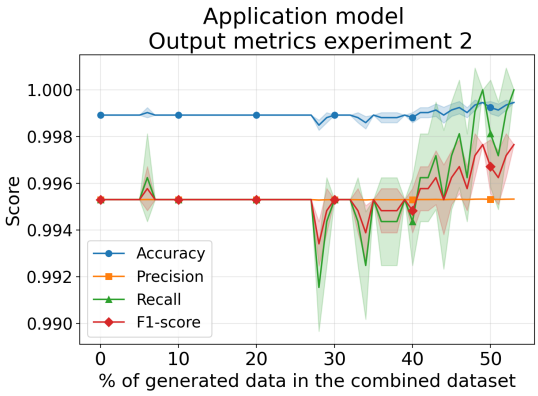
(c) The confusion matrix over different percentages of generated data in the combined training dataset of the VM in experiment 2. The mean of 5 repetitions is plotted for each type with a shadow around the mean indicating the standard deviation over the 5 repetitions.

Figure 4.8: Confusion matrices over different percentages of generated data in the combined training dataset for all three models in experiment 2.

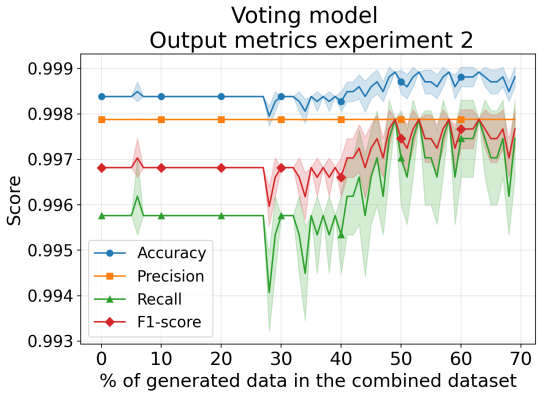
The next data that will be represented are the output metrics of all three models. These include the Accuracy, the Recall score, the Precision score and F1-score of the trained models. The output metrics for the Network Traffic- and the AM, are based on their own test sets. The output metrics of the VM are based on the final test set. These output metrics are plotted in Figure 4.9. Their deviations are again put in a table and they can be observed in Table 4.11. It can be observed that the Application and the VMs do have a slight decrease in all four output parameters starting at 28% of generated data mixed in the training datasets.



(a) The output metrics over different percentages of generated data in the combined training dataset of the NTM in experiment 2. The mean of 5 repetitions is plotted for each metric with a shadow around the mean indicating the standard deviation over the 5 repetitions.



(b) The output metrics over different percentages of generated data in the combined training dataset of the AM in experiment 2. The mean of 5 repetitions is plotted for each metric with a shadow around the mean indicating the standard deviation over the 5 repetitions.



(c) The output metrics over different percentages of generated data in the combined training dataset of the VM in experiment 2. The mean of 5 repetitions is plotted for each metric with a shadow around the mean indicating the standard deviation over the 5 repetitions.

Figure 4.9: Output metrics over different percentages of generated data in the combined training dataset for all three models in experiment 2.

Table 4.11: Percentages at which the mean of the output metrics of each model reaches outside of the range defined by the standard deviation of the baseline model parameters, including the corresponding average values and direction of change.

Model	Output parameter	Percentage	Average value at that percentage	Type of change
Network Traffic	Accuracy, Precision, Recall, F1-score	-	-	Remains within bounds
Application	Accuracy	28%	0.9985	Decrease
	Precision	28%	0.9953	Decrease
	Recall	28%	0.9915	Decrease
	F1-score	28%	0.9934	Decrease
Voting	Accuracy	28%	0.9980	Decrease
	Precision	28%	0.9979	Decrease
	Recall	28%	0.9941	Decrease
	F1-score	28%	0.9960	Decrease

The final plot contains the overall trust score of the VM. This will be displayed in Figure 4.10. The percentages from which the trust scores start to deviate outside the range of the standard deviation of the baseline experiment, are displayed in Table 4.12. It can be seen that only the trust score for test data with label 1 increases after more than 11% of generated data is mixed in the combined training datasets. The trust score for data with label 2 does change significantly as well, but this only decreases further, approaching 0.5 at high percentages. It should be mentioned that, despite the AM did not receive more generated data in its training than 53%, the plot does continue to 69%. This is due to the NTM still receiving generated data in its training set, simply because more generated samples were available. Therefore the only change in inputs from 53% onward is taken by the NTM.

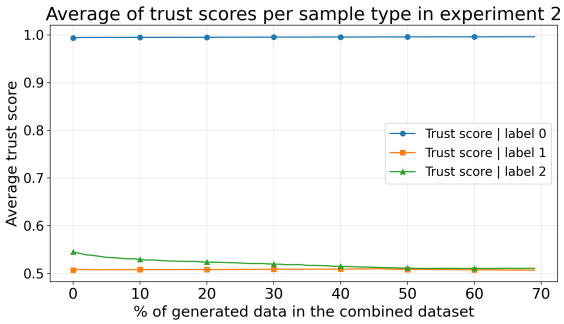


Figure 4.10: Trust score of Voting system compared to the percentage of LLM generated data in the training dataset.

Table 4.12: Percentages at which the trust score of the Voting system starts to reach outside of the range defined by the standard deviation of the baseline model trust scores.

Trust score	Percentage	Average trust score at that percentage	Type of change
Test samples with label 0	-	-	Always remains lower than baseline experiment
Test samples with label 1	11%	0.5081	Increase
Test samples with label 2	-	-	Always remains lower than baseline experiment

5 Discussion

In this chapter, the findings from background research in Chapter 2, the analysis of the design in Chapter 3 and the findings from the experiments in Chapter 4 will be discussed to provide answers to the initially posed research questions in Chapter 1. Furthermore, the limitations involved with this work will be discussed and last a brief look will be taken into what possible work still lies ahead after this thesis and what future work could consist off.

5.1. Results and problem solution

5.1.1. IDS trained with generated data

In the beginning of this thesis, three major research questions were proposed. This section will dive into finding answers to those questions. To start with question 1, where the question was ”to what extend AI-generated data can supplement or replace real-world data in training deep learning-based IDS models, it is key to observe the findings of the experiments that have been performed in Chapter 4. The two experiments were aimed at finding the influence of ever increasing percentages of AI generated data on the performance of both the application and the NTM. This question aims at finding out what happens in general when generated data is mixed in with real-recorded data and what those models then are capable of. For that it is mostly interesting to dive into the results of experiment 2.

In experiment 2, it can be observed straight away in the final trust scores of Figure 4.10 that, despite the trust score of anomalous data with label 1 slightly increasing when more generated data is mixed into the training datasets of the models, the trust score for data with label 2 actually decreases more when generated data is added in. And on top of that the average trust scores for samples with label 0 does increase. This signifies that the models have actually increased the capability to distinguish normal from anomalous data. When comparing the output metrics of the different models in Figure 4.9, it becomes clear that most of this is due to the contribution of the AM. While the output metrics of the NTM stay fairly constant, the output metrics of the AM seem to follow a capricious trend once the percentage of generated data increases. In some instances, some of the output metrics drop slightly down only for them to rise above their previous trend shortly after. In general they are improving with more generated data being added. This impact plays a direct role in the performance of the VM as can be seen in the same Figure, where the output metrics also generally increase once more than 40% of the training dataset consists of generated data. This does explain why mostly the trust score belonging to the anomalous data with label 2 is improving. This is the data that is marked anomalous by application content. It is the type of samples with either a broken temperature, humidity or PM sensor, anomalies that the AM is specifically trained to detect. This behaviour is again confirmed by observing the raw logit output in Figure 4.7. Where the logits indicate a stronger confidence in detecting the correct label as soon as more generated data is added. The logit for ’0’ increases slightly for data with label 0, as soon as it is trained with more generated data. So does the logit for ’1’ for data with label 1. While they decrease even stronger when data of the opposite label is proposed to them. This increase in distance between their two outcomes shows that the model can distinguish anomalous data better from real data when generated data is added to the training mix. With False negatives and False positive predictions

reaching almost 0 as can be observed in Figure 4.8. What could play a role here is the increase in sample size for the models. As was written by Alwosheel et al. [3], large models need large datasets to be trained on. By increasing the dataset artificially it seems the model benefitted in performance from seeing a wider variety of data while still being presented with the original recorded dataset to remain fitted to the user case, instead of degrading.

How different it is for experiment 1, where replacing real-recorded training data with generated data causes a decreased performance of all models in different amounts. The NTM seems not to be affected much at first, where its output metrics only start to drop sharply when close to a 100% of generated data is mixed in, according to Figure 4.3a. For the application and therefore the VM (that depends in a great manner from the behaviour of each single model), the story is different. The performance of this model seems to decrease almost from the start. At 10% of generated data mixed into their training sets, the AM's output metrics already leave the boundaries set by the baseline model, for them to start dropping sharply at 45% and up. Figure 4.3 could provide us with a clue to why this happens. When observing the progress of the confusion matrix of the AM, it is observed that the amount of False positives seem to rise exponentially after more than 60% of generated data is added to the training dataset (where the average false positive count actually already leaves the pre determined bounds at 39%). This to the cost of the True negative count that decreases sharply.

To explain why this might happen, it is important to look more into the data itself that is being used to train the models. After all, a deep learning model is merely forming predictions based on an abstract image of the distribution of the training data it received. In particular the AM data sparks great interest due to the performance of the AM in both experiments. In Sections 3.3.4 for the real-recorded data and 3.4.2 for the generated data, the statistics belonging to the real and generated datasets are discussed. There it was found that the real-recorded AM data proves to be of a quite constant nature. It has to be mentioned that the time of the year in which this data was recorded is usually not known for its great temperature differences, where autumn marks the crossover from hot summer days to a cooler winter. Since the air humidity is also directly dependant on the air temperature, there are no great fluctuations to be measured here as well. And with the absence of smog filled days that could occur in the summer when the wind is usually calmer, the PM counter also showed quite a constant behaviour. All this together forms a very narrow range for the model to shape itself to. Therefore, when this real-recorded data is being replaced by generated data (of which the distribution will be discussed more in Section 5.1.2 of the discussion), the shallow range in which the recorded data would fit well is all of a sudden expanded. And with this greater variance of the mixed data, samples where a broken sensor indicates a radically different value from the mean (either very high or usually for this experiment it would produce a 0) might all of a sudden be marked as normal instead. While adding the data to the original dataset as was done in experiment 2, makes the model not lose its knowledge about the real-recorded data, merely extending the precision of the barrier between normal and anomalous data.

For the NTM in experiment 1, it seems that generated data does not affect the model as much as what would be expected from the WD analysis earlier in Section 3.4.2. Only when 100% of the data is replaced by generated data and therefore no original recorded data is part of the training set any more, the model will start to consider many of the normal real-recorded samples to be anomalous. With the model starting to fall slightly out of its bounds by 45% or more of the combined dataset being filled with generated data. The reason why it might be relatively tolerant to the introduction of generated data could be due to the volatility of NTT parameters. Opposite to the AM data with its relatively constant measurements, the NTT data has relative larger variances due to the nature of the technology responsible for the packet delivery. It seems that the LSTM architecture therefore reacts with more tolerance on data with a larger variance.

One of the research papers that served as inspiration for this thesis [29] suggests something similar when generating imagery with deep learning based models in order to feed these back into the training of the same model. In this paper it is suggested that the collapse of the model could be prevented as long as in each cycle a certain ratio of realistic data is combined with the artificially generated dataset. This poses a stronger support for the methods of experiment 2, to add generated data on top of real-recorded samples instead of removing the real-recorded data.

To summarise this analysis, it seems that the LSTM architecture used in this thesis is in general adjusting very well to the variance of the training data it is offered. Therefore, it really depends on the distribution of the data that is used to train it, how tolerant these models are to the influence of LLM generated data once real-recorded data is being replaced. With more fluctuating data the impact of adding generated data to the training set is relatively marginal. With data that has a less variant distribution like temperature readings or PM counts, the model fits itself tighter to the distribution, prone to making more mistakes. By adding the generated data to the original recorded dataset the performance of the models however is improved. The increased dataset size could be the reason behind this improvement.

5.1.2. Quality of synthetic data

Question 2 put forward the inquiry to 'Whether synthetic data can accurately capture the temporal and contextual dependencies present in real sensor measurements'. This has been researched for most part in the data analysis of Section 3.3.4 and Section 3.4.2. This question mostly focusses on replacing the real-recorded sensor measurements for the AM with LLM generated data. When observing the generated data and comparing this with the real-recorded data, it seems that the LLM has managed well to capture the distribution of the environmental measurements, the temperatures were slightly lower than those of the recorded dataset and the humidity therefore slightly higher, but not to what seems out of the ordinary for these places in the Netherlands when considering their current climate statistics.

For the PM values, slightly higher counts were generated by the LLM. Although this does reflect a more global representation of the average fine dust particle count in the air and the counts were still below the limits of what is considered abnormal to the air quality in the Netherlands, it was significantly different from the average counts that were recorded through the weather stations. The blame cannot be found in lack of online references for the LLM to base its decisions on. Altough the few sources that specifically measure the particle count in the Netherlands fall into insignificance compared to the vast amount of sources available to the LLM for its training, the information is there. This could be one of the factors explaining why the AM lost some of its performance with the replacement of real-recorded data by generated data. An indicator suggesting strongly that the recorded data could be replaced by the generated data, however, is the WD that was calculated between the two datasets. With the WD between the generated and recorded datasets being significantly lower than the WDs calculated among the recorded dataset itself. However, most of the high WDs that were found when performing these calculations are most likely due to the anomalous behaviour of the weather station in Den Hoorn. Since this was used to produce anomalous data for the AM dataset and for the NTT dataset, it is shorter on normal samples compared to the other weather stations. This could be the reason why Den Hoorn appeared in most of the 'largest' WDs.

Although the statistics do not show a clear indication that the generated AM data is unsuitable for training the AM, it has been concluded in practice that replacing the real-recorded dataset with generated data does not improve the model. Only when supplementing the recorded data set with generated data was an improvement (albeit marginal) was detected. So, it seems that synthetic data can indeed accurately capture the temporal and contextual dependencies present in real sensor measurements to a certain limit. It will not be able to fully replace the real recordings when it comes to generating an application deep-learning-based IDS.

5.1.3. Performance over percentage

Last but not least, question 3 considered the ratio of real-to-synthetic data and how it affects model performance. It has already been partially made clear in Section 5.1.1 that the influence of generated data depended greatly on the fact if it was added on top of the recorded dataset, or if it replaced the original recorded dataset, with the latter eventually causing performance degradation. Initially an estimation or prediction was made in Section 3.4.2, that for the AM data, virtually all the recorded data would be replaceable by the generated data without causing performance degradation. For the NTM, it was predicted that the model could experience performance degradation when more than 13% of the combined training dataset would consist of LLM generated data. It turned out when reviewing the results in Section 4.3, that the WD distance does not provide enough information to make a prediction

on the limits of how much generated data can be mixed in. The AM data turned out to be far less replaceable than predicted, whereas the NTM only suffered from performance degradation as soon as all the real-recorded data was replaced with artificially generated data. So then the question remains: what is the impact of the ratio of real-to-synthetic data on model performance?

This is a question that can partially be answered. It is observed that depending on how the generated data is added, the model can either increase in performance slightly (for as far as this could improve from the base model) or decrease in performance. However, it did not appear that making 10% or less of the combined dataset by replacing or adding, had any significant influence on the model's performance. It is suggested however, that adding generated data on top of real-recorded samples will have preference over replacing recorded data. And that real-recorded data will still remain of big importance for the performance of deep learning based IDS. Inserting artificial data can make improvements, but have to be crafted with great consideration.

5.2. Limitations

This thesis has experienced some limiting factors that reduced the generality of the researched topic to a certain amount. To start, the context that was chosen for this thesis (the environmental measurement application) has shown to be a relatively simple context to train an IDS for. The data that was recorded is of a relatively constant nature and therefore showed that an LSTM model can fit itself with ease to the dataset that it delivers. This means that, although this allows one to have a more thorough understanding of the type of data that was recorded, the possibility for improving a model is taken away. It is a topic of great interest to investigate if LLM generated data could have caused a more significant improvement of the baseline models when it is added on top of real-recorded data. But with only 1 registered false negative and 2 false positives, there is not much space for improvement any more.

On top of this, the scope of research is also quite limited. Since no significant previous research was found on training deep learning based IDS' with artificially generated data, a certain baseline had to be set to investigate the general impact of the generated data on these model's performances. However, to restrict the scope of this thesis, it is therefore not suitable to try different model architectures or different LLM's already. This could serve as future work however.

Another limiting factor is that the performance of the deep learning models is greatly dependant on the content of the data. It could be that with different applications, the impact of generated data could be very different. Therefore it is not advised to use conclusions from this thesis as a general method for replacing recorded dataset with LLM generated data. It serves merely as a basis to show that with some applications it can indeed replenish AM data and possibly improve model performance with this. But more research will be needed to transform this into a general statement.

Last but not least, the NTT data could have been more extensive. The recording software on the server and the environmental measurement application did not allow for more thorough recording of different kinds of NTT data. This would again extend the scope of this thesis in such a way that the core topic would fall into insignificance through the jungle of different set-ups. This again argues in favour of a comparative research with different contexts to create a more general statement.

5.3. Future Works

As mentioned in the previous section, there are many topics of interest flowing forward from this thesis. While this thesis serves as an opening to research on deep-learning IDS trained with artificially generated data, it proves to be very context specific. Therefore different set-ups and contexts should be tried and tested.

For example, the use of auto encoders in combination with the LSTM to remove the need for labelled training data could sincerely improve the performance of IDS. It would serve a great interest to investigate the behaviour of such an IDS when trained partially with LLM generated data.

More complex contexts should also be researched. While this thesis focused merely on the basis and what the influence of LLM generated data is on a relatively simple IDS, more complex data types with more features and more space for error would allow research on the possible improvements that could be made on an IDS when generated data is added to the training dataset.

Furthermore, different LLMs should also be considered for generating the training datasets. This thesis has focused on one of them, to keep the scope limited. However, it should be considered that the accuracy of the generated data relates heavily on the LLM it is generated by. Therefore it is advised to perform a comparative study on generated datasets by LLM's in the context of deep-learning based IDS.

6

Conclusion

This thesis serves as an opening to the research topic of the use of LLM generated datasets into the training of deep-learning based IDS. In the context of a small distributed environmental measurement application, AM data in the form of temperature, air humidity and fine particle counts have been recorded over a period of a week. Along with this AM data, the NTT data belonging to the http traffic between the collecting server and the measurement stations has been collected and recorded using Zeek. These datasets have been used to train multiple deep learning models with the Long-Short-Term Memory architecture to establish two base models. One that is used to detect anomalies on the AM data and one that served to detect anomalies on the incoming network traffic. Following this, and using a pre-established prompting strategy, a widely available publicly accessible LLM was used to generate synthetic datasets containing application and NTT data. These have been combined with the recorded datasets in two different experiments to investigate the impact of artificially generate data on the performance of the base models. The Wasserstein distance has been used as an estimator to predict with what percentage of generated data, a decline in performance of these two models would occur.

It has been observed that, when replacing the recorded dataset progressively with the generated data samples, the AM experiences a performance decline starting with 10% of the combined dataset consisting of generated data samples. However, when adding the generated data to a combined dataset without replacing the original recorded data, it has been observed that the performance of the base models is largely preserved, even slightly improved. This has been argued to be highly dependant, however, on the distribution of the recorded datasets, where it seems that data with in general a higher variant nature is less affected by the replacement by artificially generated data. The Wasserstein, albeit useful to compare datasets on their distributions, turned out not to be a significant predictor on the influence of adding generated data samples to the combined dataset and the from this arising model performance.

It is advised for future research to investigate different contexts with different data types on the performance of deep-learning based IDS when artificially generated data is added to the training set. More complex applications allow for the investigation on possible improvement of such models by introducing artificially generated data, if the baseline performance does not reach a performance limit. Furthermore, different LLMs should be tried and tested in a comparative study to discover the impact of these on the quality of the generated data and therefore the influence on an IDS.

This thesis concludes with a partial victory: that the incorporation of artificially generated application and NTT data can, within certain constraints as identified in the findings, lead to modest performance improvements in deep-learning-based IDS models.

Bibliography

[1] Abeer Al-Ajlan and Mourad Ykhlef. *A Review of Generative Adversarial Networks for Intrusion Detection Systems: Advances, Challenges, and Future Directions*. 2024.

[2] Sina Alemohammad et al. “Self-Consuming Generative Models Go MAD”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2024. URL: https://openreview.net/forum?id=ShjMHfmPs0.

[3] Ahmad Alwosheel, Sander van Cranenburgh, and Caspar G. Chorus. “Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis”. In: *Journal of Choice Modelling* 28 (2018), pp. 167–182. ISSN: 1755-5345. DOI: https://doi.org/10.1016/j.jocm.2018.07.002. URL: https://www.sciencedirect.com/science/article/pii/S1755534518300058.

[4] Shai Ben-David et al. “A Theory of Learning from Different Domains”. In: *Machine Learning*. Springer, 2010, pp. 151–175. DOI: 10.1007/s10994-009-5152-4.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[6] Canonical Ltd. *Install Ubuntu on a Raspberry Pi*. Accessed: 2025-11-17. Canonical Ltd. 2025. URL: https://ubuntu.com/download/raspberry-pi.

[7] DeepSeek. *Deepseek*. URL: https://chat.deepseek.com/.

[8] J. A. Delgado-Soto et al. “Towards Realistic Network Traffic Conversations Generated with Large Language Models”. In: *Computer Networks* (2025). Online first. URL: https://www.sciencedirect.com/science/article/pii/S1389128625002762.

[9] Hamouda Djallel et al. “Advancing Cybersecurity with LLMs: A Comprehensive Review of Intrusion Detection Systems and Emerging Applications”. In: *Proceedings of the International Conference on Informatics and Applied Mathematics (IAM 2024)*. Vol. 3922. CEUR Workshop Proceedings. Accessed: 2025-11-02. Guelma, Algeria: CEUR-WS.org, Dec. 2024. URL: https://ceur-ws.org/Vol-3922/paper7.pdf.

[10] Wesley Eddy. *TCP SYN Flooding Attacks and Common Mitigations*. RFC 4987. Aug. 2007. DOI: 10.17487/RFC4987. URL: https://www.rfc-editor.org/info/rfc4987.

[11] Mohamed Amine Ferrag et al. “Deep Learning for Cyber Security Intrusion Detection: Approaches, Datasets, and Comparative Study”. In: *Journal of Information Security and Applications* 50 (2020), p. 102419. DOI: 10.1016/j.jisa.2019.102419. URL: https://doi.org/10.1016/j.jisa.2019.102419.

[12] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. “Learning to Forget: Continual Prediction with LSTM”. In: *Neural Computation* 12.10 (2000), pp. 2451–2471. DOI: 10.1162/089976600300015015.

[13] Ian J. Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 27. 2014. URL: https://arxiv.org/abs/1406.2661.

[14] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[15] Zhijing Ji et al. “Survey of Hallucination in Natural Language Generation”. In: *ACM Computing Surveys* (2023). DOI: 10.1145/3571730.

[16] Feng Jiang et al. “Deep Learning Based Multi-Channel Intelligent Attack Detection for Data Security”. In: *IEEE Transactions on Sustainable Computing* 5.2 (2020), pp. 204–212. DOI: 10.1109/TSUSC.2018.2793284.

[17] Ansam Khraisat et al. “Survey of intrusion detection systems: techniques, datasets and challenges”. In: *Cybersecurity* 2.1 (2019). ISSN: 2523-3246. DOI: 10.1186/s42400-019-0038-7.

[18] Jan Lánský et al. “Deep Learning-Based Intrusion Detection Systems: A Systematic Review”. In: *IEEE Access* 9 (2021), pp. 101574–101599. URL: https://api.semanticscholar.org/CorpusID:236479767.

[19] Yan Liu et al. “A Review of Rule Learning Based Intrusion Detection Systems and Their Prospects in Smart Grids”. In: *IEEE Access* 9 (2021). Accessed: 2025-11-02, pp. 29605–29622. DOI: 10.1109/ACCESS.2021.3058584. URL: https://ieeexplore.ieee.org/document/9351272.

[20] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2018.

[21] OpenAI. *ChatGPT*. URL: https://openai.com/index/chatgpt/.

[22] Gabriel Peyré and Marco Cuturi. *Computational Optimal Transport*. Now Publishers, 2019. DOI: 10.1561/2200000073.

[23] Raspberry Pi Ltd. *Raspberry Pi Documentation*. Accessed: 2025-11-17. Raspberry Pi Ltd. 2025. URL: https://www.raspberrypi.com/documentation/.

[24] Michael Ring. “A survey of network□based intrusion detection data sets”. In: *Computers Security* (2019). Accessed: YYYY-MM-DD.

[25] Robin Rombach et al. “High-Resolution Image Synthesis with Latent Diffusion Models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 10684–10695. URL: https://arxiv.org/abs/2112.10752.

[26] Seeed Studio Co., Ltd. *Grove – Laser PM2.5 Sensor (HM3301)*. Accessed: 2025-11-17. Seeed Studio Co., Ltd. 2023. URL: https://wiki.seeedstudio.com/Grove-Laser_PM2.5_Sensor-HM3301/.

[27] Seeed Studio Co., Ltd. *Grove – Temperature & Humidity Sensor (DHT20)*. Accessed: 2025-11-17. Seeed Studio Co., Ltd. 2025. URL: https://wiki.seeedstudio.com/Grove-Temperature-Humidity-Sensor-DH20/.

[28] Seeed Studio Co., Ltd. *Seeed Studio Wiki*. Accessed: 2025-11-17. Seeed Studio Co., Ltd. 2025. URL: https://wiki.seeedstudio.com/.

[29] Ilia Shumailov et al. “AI Models Collapse When Trained on Recursively Generated Data”. In: *Nature* 631.8024 (2024), pp. 755–760. DOI: 10.1038/s41586-024-07566-y.

[30] A. Smajić et al. “Large Language Models for Structured and Semi-Structured Data, Recommender Systems and Knowledge Base Engineering: A Survey of Recent Techniques and Architectures”. In: *Electronics* 14.15 (2025), p. 3153. DOI: 10.3390/electronics14153153.

[31] Salvatore Stolfo et al. *KDD Cup 1999 Data: The Third International Knowledge Discovery and Data Mining Tools Competition — Data Set*. https://www.kdd.org/kdd-cup/view/kdd-cup-1999. Accessed: 2025-11-15. 1999.

[32] T. A. Tang et al. “Deep Recurrent Neural Network for Intrusion Detection in SDN-based Networks”. In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 202–206. DOI: 10.1109/NETSOFT.2018.8459910. URL: https://ieeexplore.ieee.org/document/8459910.

[33] Mahbod Tavallaee et al. *NSL-KDD dataset for network-based intrusion detection systems*. http://nsl.cs.unb.ca/NSL-KDD/. Accessed: 2024-10-01. 2009.

[34] Alexandre Tsybakov. “Optimal Aggregation of Classifiers in Statistical Learning”. In: *Annals of Statistics* 32.1 (2004), pp. 135–166.

[35] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems (NeurIPS) 30*. 2017, pp. 5998–6008. URL: https://arxiv.org/abs/1706.03762.

[36] The KDD Cup 1999 Data Collection Working-Group. *KDD Cup 1999: Computer network intrusion detection (data set)*. https://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data. Accessed: 2025-11-15. 1999.

[37] Zhiwei Xu et al. *Deep Learning-based Intrusion Detection Systems: A Survey*. 2025. arXiv: 2504.07839 [cs.CR]. URL: https://arxiv.org/abs/2504.07839.

[38] Chuanlong Yin et al. “A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks”. In: *IEEE Access* 5 (2017), pp. 21954–21961. DOI: 10.1109/ACCESS.2017.2762418. URL: https://ieeexplore.ieee.org/document/8091083.

[39] Ya Zhang, Ravie Chandren Muniyandi, and Faizan Qamar. “A Review of Deep Learning Applications in Intrusion Detection Systems: Overcoming Challenges in Spatiotemporal Feature Extraction and Data Imbalance”. In: *Applied Sciences* 15.3 (2025). ISSN: 2076-3417. DOI: 10.3390/app15031552. URL: https://www.mdpi.com/2076-3417/15/3/1552.