12-2023

Adversarially Reweighted Sequence Anomaly Detection With Limited Log Data

Kevin Vulcano
Utah State University, kevin.vulcano@usu.edu

ADVERSARIALLY REWEIGHTED SEQUENCE ANOMALY DETECTION WITH LIMITED LOG DATA

by

Kevin Vulcano

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Shuhan Yuan, Ph.D. Major Professor	Mario Harper, Ph.D. Committee Member
Curtis Dyreson, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2023

ABSTRACT

Adversarially Reweighted Sequence Anomaly Detection
with Limited Log Data

by

Kevin Vulcano, Master of Science
Utah State University, 2023

Major Professor: Shuhan Yuan, Ph.D.
Department: Computer Science

The field of log sequence anomaly detection is of paramount significance, finding wide-ranging applications in areas such as cybersecurity, network surveillance, industrial processes, and financial transaction monitoring. This thesis introduces AdvSVDD, a deep learning model for sequence anomaly detection. AdvSVDD, based on the concept of Deep Support Vector Data Description (Deep SVDD), excels in the identification of anomalies within log sequences and enhances performance under limited training data through the integration of Adversarial Reweighted Learning (ARL). By employing the Deep SVDD technique to map normal log sequences into a hypersphere and leveraging the loss amplification effects of Adversarial Reweighted Learning (ARL), AdvSVDD can be effectively trained with normal log sequences, enhancing its robustness in detecting anomalies. Experiments conducted on the BlueGene/L (BG/L) and Thunderbird supercomputer datasets show that AdvSVDD can achieve higher performance than traditional machine learning and deep learning approaches, including the foundational Deep SVDD framework. The models were evaluated on five metrics: Precision, Recall, F1-Score, ROC AUC and PR AUC. Additional studies demonstrate the effectiveness of AdvSVDD under limited training data and provide insight into the role of the adversarial component.

(65 pages)

PUBLIC ABSTRACT

Adversarially Reweighted Sequence Anomaly Detection
with Limited Log Data
Kevin Vulcano

In the realm of safeguarding digital systems, the ability to detect anomalies in log sequences is paramount, with applications spanning cybersecurity, network surveillance, and financial transaction monitoring. This thesis presents AdvSVDD, a sophisticated deep learning model designed for sequence anomaly detection. Built upon the foundation of Deep Support Vector Data Description (Deep SVDD), AdvSVDD stands out by incorporating Adversarial Reweighted Learning (ARL) to enhance its performance, particularly when confronted with limited training data. By leveraging the Deep SVDD technique to map normal log sequences into a hypersphere and harnessing the amplification effects of Adversarial Reweighted Learning, AdvSVDD demonstrates remarkable efficacy in anomaly detection. Empirical evaluations on the BlueGene/L (BG/L) and Thunderbird supercomputer datasets showcase AdvSVDD's superiority over conventional machine learning and deep learning approaches, including the foundational Deep SVDD framework. Performance metrics such as Precision, Recall, F1-Score, ROC AUC, and PR AUC attest to its proficiency. Furthermore, the study emphasizes AdvSVDD's effectiveness under constrained training data and offers valuable insights into the role of adversarial component has in the enhancement of anomaly detection.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ARL	Adversarially Reweighted Learning
BERT	Bidirectional Encoder Representations from Transformers
BG/L	BlueGene/L Supercomputer System
Drain	Depth tree based online log parsing
FN	False Negative
FP	False Positive
GAN	Generative Adversarial Network
GRU	Gated Recurrent Units
KNN	K Nearest Neighbor
LSTM	Long short-term memory
MLKP	Masked LogKey Prediction
MSE	Mean Squared Error
MVP-tree	Multi-Vantage Point tree
NCD	Normalized Compression Distance
NLP	Natural Language Processing
OCSVM	One-Class SVM
PCA	Principal Component Analysis
PR AUC	Precision Recall Area Under the Curve
RBF	Radial Basis Function
RNN	Recurrent Neural Network
ROC AUC	Receiver Operating Characteristic Area Under Curve
Spell	Streaming Parser for Event Logs using LCS
SVDD	Support Vector Data Description
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
VHM	Volume of Hypersphere Minimization

CHAPTER 1

Introduction

1.1 Background

In this section, an overview of the key concepts related to anomaly detection and log anomaly detection is presented. The section begins with a discussion of the importance of anomaly detection in real-world applications across various domains. Subsequently, the section explores different types of anomalies and their classifications. Finally, the field of log anomaly detection is introduced, along with an outline of the challenges associated with it.

1.1.1 Anomaly Detection

Anomaly detection, also known as outlier detection, is a crucial task in the analysis of real-world datasets. Its primary objective is to identify instances that deviate significantly from the norm, often referred to as anomalies. These anomalies may result from data errors or, in some cases, reveal novel and previously unknown underlying patterns or processes. As described by Hawkins, an outlier is an observation that deviates so significantly from other observations as to arouse suspicion that it was generated by a different mechanism [1]. Anomaly detection plays a pivotal role across various domains, highlighting its significance in today's data-driven applications. Successful anomaly detection algorithms find widespread application in diverse fields, where they are central to maintaining data integrity, ensuring security, and uncovering invaluable insights. This technology serves multiple purposes, including fraud detection in financial services, quality control in manufacturing, network security in cybersecurity, diagnosis in healthcare, environmental and industrial process monitoring.

Applications of Anomaly Detection

Fraud Detection In the realm of financial services, anomaly detection serves as a guardian against fraudulent activities. By inspecting transactions and account behaviors for deviations from typical spending or usage patterns, it aids in the identification of potentially fraudulent activities [2].

Quality Control In manufacturing, ensuring product quality is essential. Anomaly detection is employed to monitor manufacturing processes, identifying deviations from product specifications or performance [3]. This ensures identification and rectification of subpar or faulty products, maintaining high standards and minimizing waste.

Network Security Cybersecurity relies on anomaly detection to safeguard networks and systems against malicious intrusions. By continuously monitoring network traffic and system behavior, any anomalous activities indicative of cyberattacks or breaches can be detected in real-time [4]. This proactive approach fortifies digital defenses and protects sensitive data.

Healthcare Anomaly detection plays a pivotal role in healthcare, aiding in the diagnosis of rare diseases and the detection of medical anomalies. Medical professionals rely on anomaly detection to identify irregularities in patient data, such as vital signs, lab results, or medical images. This assists in the early detection of health issues and ensures timely interventions, potentially saving lives. [5]

Environmental Monitoring Anomaly detection extends its utilization to environmental sciences, where it is instrumental in monitoring environmental parameters [6]. By identifying unusual changes in climate data, pollution levels, or ecosystem behavior, scientists can better understand environmental shifts and take preventive actions against natural disasters or ecological crises.

Anomaly Types

Although there are many flavors of anomaly detection data, they can be categorized as sequential and non-sequential [7]. Image data is a type of non-sequential data, whereas

video, text and speech are considered sequential data.

In addition, anomalies can be classified into three distinct categories: point or simple anomalies, and two types of complex anomalies—contextual anomalies and collective anomalies [8]. Point anomalies, the simplest form of anomalies, represent outliers that significantly deviate from the expected or normal behavior.

Complex anomalies, such as contextual and collective anomalies, involve intricate patterns in the data [9]. Contextual anomalies are data points that may appear normal in isolation but are deemed unusual within specific conditions or context, relying on contextual information for identification. Collective anomalies, involve abnormal behavior exhibited by a group of data points as a whole, rather than by an individual data point.

Figure 1.1 provides a demonstration of all three anomaly types. Point anomalies are presented by highlighting a specific instances of anomalous spending within a currency dataset. Collective anomalies are illustrated by showcasing unusual repetitive transactions that constitute abnormal behavior. Contextual anomalies are emphasized using four time series plots. The top two depict typical patterns of house and (Photovoltaic) PV load, while the bottom two represent these patterns during an attack scenario. Notably, all data point values within these time series could resemble those of normal behavior, but the context surrounding them is critical for identifying the series as anomalous.

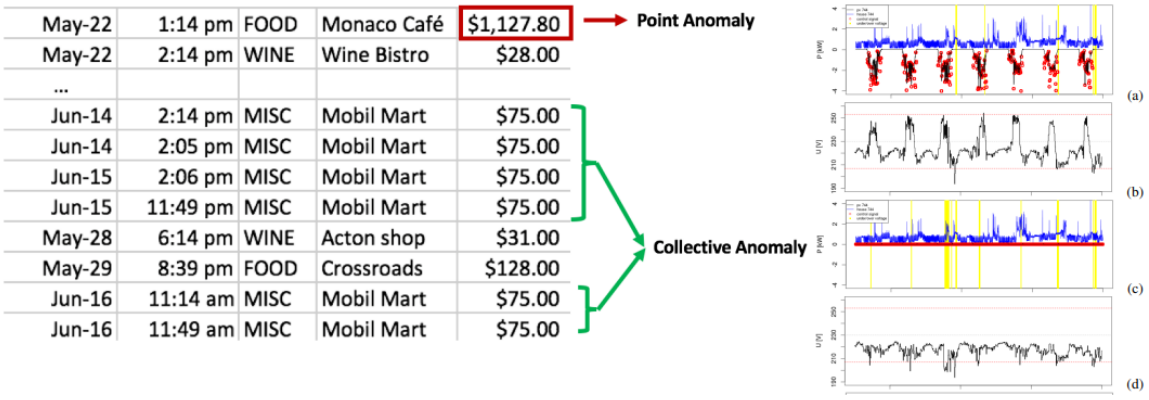

Fig. 1.1: Left: The image displays an example highlighting the anomaly categorized as a point anomaly in red, and the collective anomaly example is illustrated in green [8]. Right: The image displays an example of a contextual anomaly [10].

1.1.2 Log Anomaly Detection

Log anomaly detection holds significant importance in the fields of data analysis and cybersecurity. It revolves around the identification of unusual patterns or irregularities within log data, generated by computer systems or applications. These logs serve as valuable repositories of information for tasks such as system monitoring, troubleshooting, and security analysis. However, detecting anomalies within the vast volume of log entries, which may indicate security breaches, system failures, or operational issues, poses a formidable challenge.

The primary objective of log anomaly detection is the automated and systematic identification and flagging of these anomalies within log data. This process relies on various techniques from statistics and machine learning to differentiate abnormal log entries from those considered normal. Effective log anomaly detection plays a critical role in safeguarding the security and reliability of computer systems and networks.

Categories of Log Sequence Anomalies

In log sequence anomaly detection, anomalies can be categorized into three distinct categories: Execution Order, Operation, and Incomplete anomalies [11], each addressing a specific aspect of abnormal log behavior.

Consider a typical log sequence in a web server application:

$$[A1, D1, A2, D2, A3, D3]$$

In this non-anomalous log sequence, there is a series of events represented by logs. These logs include user authentication logs A followed by data access logs D, and they occur in the expected order. This sequential pattern aligns with the normal behavior of the system, where users first authenticate A and then access data D. Table 1.1 demonstrates logs for the examples.

Execution Order Anomaly occurs when a sequence of logs contains normal logs, but their execution order deviates from the expected pattern. This type of anomaly is

Table 1.1: Log Types and Examples

Log Type	**Example Log**
A (Authentication)	[TIMESTAMP]: [USER] authenticated
D (Data Access)	[TIMESTAMP]: [USER] accessed file [FILENAME]
P (Prohibited)	[TIMESTAMP]: Unauthorized access attempt by [USER]

characterized by a disruption in the typical workflow or sequence of events within the log data. Consider the following log sequence:

$$[A1, D1, A2, D2, D3, A3]$$

In this case, the execution order anomaly occurs because the expected order of authentication A followed by data access D is disrupted. In the example, $D3$ is logged before $A3$.

Operation Anomaly is characterized by logs appearing in the log sequence when they shouldn't, regardless of whether these logs are normal or abnormal. It signifies that certain actions or events have occurred out of place within the sequence. Continuing with the same log types, consider the following log sequence:

$$[A1, A2, D1, D2, P1, A3, D3]$$

In this sequence, a prohibited action log $P1$ appears in the sequence, which is unexpected. The presence of this action within the sequence classifies it as an operational anomaly.

Incomplete Anomaly is determined when the logical correlation among logs in a sequence is incomplete. It suggests that the contextual information necessary to fully understand the sequence of events is missing. Consider the following log sequence:

$$[A1, A2, D1, A3, D3]$$

In this sequence, all log types (authentication and data access) are present, but an

incomplete anomaly occurs because the data access $D2$ is missing. The full sequence of events cannot be understood without it.

1.1.3 Challenges in Log Sequence Anomaly Detection

In the field of log sequence anomaly detection, many challenges are commonly encountered due to the nature of the data:

- High-Dimensional Data: Log sequences often involve a high number of dimensions, making it challenging to extract meaningful patterns and detect anomalies effectively.
- Noise: The presence of noise in log data can complicate the task of distinguishing true anomalies from random fluctuations.
- Imbalanced Datasets: Imbalanced datasets, where normal log sequences significantly outnumber anomalous ones, can lead to biased models and reduced anomaly detection accuracy [8]. As a result, the majority of log sequence anomaly detection models are primarily trained using normal samples.
- Limited Training Data: Working with a relatively small set of training data can hinder the model's ability to generalize effectively to unseen sequences.
- Temporal Dependencies: Log sequences often exhibit temporal dependencies, meaning that the order of the entries matter. Capturing these dependencies accurately is crucial for effective anomaly detection.

1.2 Motivation

The motivation for this thesis rests upon two fundamental pillars: the recognition of an existing limitation within existing log anomaly detection and the innovative paradigm of adversarial learning. The first pillar pertains to an observed limitation in current log sequence anomaly detection. The second pillar, represented by adversarial learning, introduces a promising approach within the field of anomaly detection, further reinforcing the motivation for this research.

1.2.1 Limitation of Existing Log Anomaly Detection Approaches

A significant limitation in existing sequential anomaly detection approaches is their vulnerability to overfitting when trained with limited data. Deep learning models, commonly used for this task, are capable of capturing complex patterns. However, in scenarios with insufficient training data, these models may memorize noise or idiosyncrasies in the training set, which adversely affects their ability to generalize to new, unseen data. Such limitations can impede the model's ability to generalize effectively. This project was undertaken to address and overcome this specific challenge.

1.2.2 Adversarial Learning in Anomaly Detection

Utilizing adversarial models to augment learner models represents a relatively novel and innovative approach within the realm of machine learning. This approach involves the collaboration of two key models: the learner model and the adversarial model, each with distinct roles and responsibilities.

Learner Model

The learner model serves as the primary model designed to perform a specific machine learning task. It may be assigned with tasks such as image recognition, natural language processing, or anomaly detection. The learner model's objective is to learn patterns, features, or representations from the data to make accurate predictions or classifications. In essence, it is the model that we aim to improve and make more resilient through the adversarial approach.

Adversarial Model

The adversarial model, on the other hand, plays a unique and crucial role. Its primary purpose is to act as an adversary to the learner model. It is designed to probe, challenge, and find weaknesses in the learner model's performance. Through iterative training and various techniques, the adversarial model attempts to craft data inputs that can mislead or confuse the learner model. By doing so, it helps reveal vulnerabilities in the learner model's

decision boundaries, enhancing the model's robustness and generalization capabilities. Essentially, the adversarial model simulates potential attacks or adversarial scenarios, pushing the learner model to become more resilient and less prone to making erroneous predictions or classifications.

Applications of Adversarial Models in Anomaly Detection

Adversarial approaches have demonstrated their effectiveness in various domains, including image recognition [12], natural language processing [13], reinforcement learning [14], and within the realm of anomaly detection [15–17]. In all these applications, Generative Adversarial Networks (GANs) play a pivotal role. GANs train an adversarial model with a generative design, in which they learn to produce erroneous data deliberately, exploiting vulnerabilities in the learner model [18]. This process allows the learner model to focus on and improve upon these identified weaknesses, thereby enhancing its performance.

However, GANs are not the sole adversarial approach for enhancing learner models. Adversarially Reweighted Learning (ARL) employs an adversary model to determine weights specifically aimed at amplifying the influence of challenging samples, those that the learner model struggles to comprehend [19]. The utilization of adversarial models, particularly ARL, in the domain of log sequence anomaly detection remains relatively unexplored. This underexplored area presents an enticing and motivating challenge.

1.3 Thesis Organization

This thesis comprises several chapters, each addressing specific aspects of the research. The following sections provide an overview of the remaining content and structure of this thesis.

Chapter 2: Related Work

In this chapter, the existing literature and research related to the topic of anomaly detection and machine learning approaches are explored. The related work is categorized

into traditional approaches, including supervised and unsupervised machine learning, and deep learning approaches, encompassing RNN-based models and Transformer-based models.

Chapter 3: Preliminary

This chapter establishes the foundation for the research by describing the essential concepts and techniques used throughout the thesis. It covers log preprocessing techniques, introduces the Deep Support Vector Data Description (Deep SVDD) method, and presents the Adversarially Reweighted Learning (ARL) framework.

Chapter 4: AdvSVDD

In this chapter, the architecture of the proposed AdvSVDD model is discussed. The components, including the anomaly detector and the adversary, are elaborated upon, along with an explanation of the objective function guiding the model's training. Additionally, insights into the training details and strategies employed in AdvSVDD are provided.

Chapter 5: Experiments

This chapter presents the empirical evaluation of AdvSVDD. The experimental setup, including datasets, baselines, evaluation metrics, and implementation specifics, is detailed. Subsequently, the experimental results are analyzed and discussed, addressing overall performance, ablation studies, and sensitivity analyses.

Chapter 6: Conclusion

The final chapter summarizes the key findings of this thesis. The implications of the research and potential avenues for future work in the field of anomaly detection using adversarial learning are discussed.

CHAPTER 2

Related Work

2.1 Traditional Approaches

This section explores traditional approaches to log anomaly detection, covering both Supervised Machine Learning and Unsupervised Machine Learning. Additionally, specific algorithms within each category will be discussed, including the following methods: Decision Trees, K-Nearest Neighbors (KNN), Support-Vector Machines (SVM), Principal Component Analysis (PCA), One-Class Support Vector Machine (OCSVM), and LogCluster.

2.1.1 Supervised Machine Learning

Supervised machine learning algorithms algorithm train on labeled data to acquire predictive capabilities. The algorithm derives insights from the training data, where each input is associated with an accurate outcome, known as a label. The objective of supervised learning revolves around developing a connection between the input data and their corresponding label. Common supervised approaches for anomaly detection are Decision Tree, KNN and SVM [20–22].

Decision Tree

The decision tree algorithm is a machine learning approach that constructs a hierarchical tree structure using input data from a training set and their associated labels. At each step, the algorithm selects the feature that maximizes information gain, a metric that quantifies the reduction in uncertainty when partitioning data based on a specific feature.

This process results in a hierarchical structure where each node represents a decision based on a feature, leading to accurate predictions. An illustrative example of a decision tree in log anomaly detection can be found in Figure 2.1. Each node in the tree represents a condition and the amount of samples that it will split. For example, the root node reports

20 samples, and after filtering the samples based on the condition event 2 = 5, it splits the data into two. The data that met the condition were a total of 12, where the child node reveals the condition for the subsequent splitting condition.

To prevent overfitting, decision trees use stopping criteria such as maximum tree depth. Decision trees effectively handle both categorical and numerical features as well as interpretability through their decision-making process and visualization. Additionally, they can be integrated into ensemble methods such as Random Forests, which utilize multiple decision trees to enhance performance.

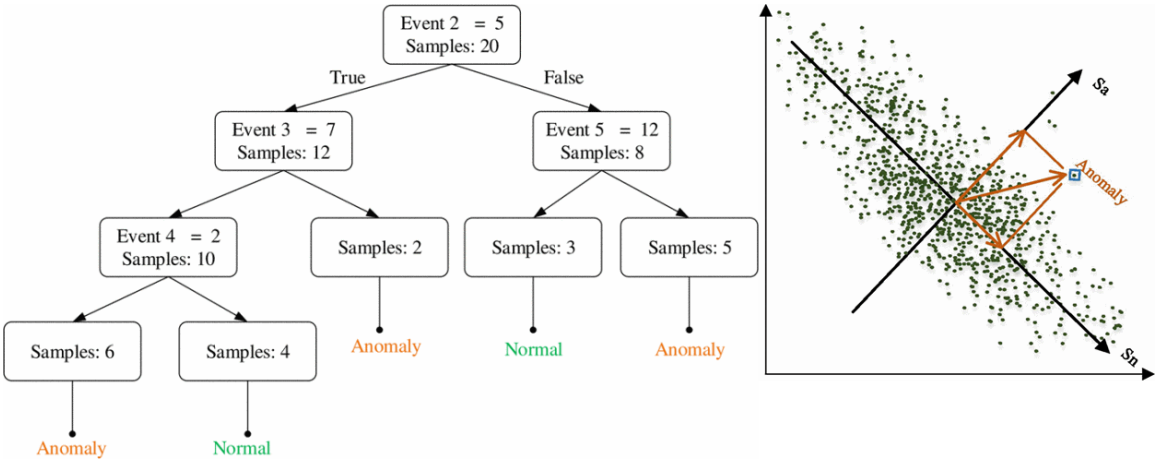

Fig. 2.1: Example of log anomaly detection using decision tree (left) and PCA (right) [23].

K-Nearest Neighbor

KNN operates by classifying data points based on the labels of their 'k' nearest neighbors, determined using distance metrics like Euclidean or Manhattan distance. KNN's strength lies in its simplicity and adaptability to local data structures [24].

However, selecting the appropriate 'k' value is a critical consideration. Smaller 'k' values yield more sensitive but potentially noisy results, while larger 'k' values can lead to oversmoothed decision boundaries. Effective hyperparameter tuning, often through techniques like cross-validation, is essential to optimize KNN for a specific anomaly detection task.

The KNN algorithm, commonly employed for anomaly detection, has been a pivotal component in a recent method for Arrhythmia detection in ECG data. It was utilized for an initial data analysis, followed by the integration of a Decision Tree for final classification [25]. The KNN algorithm has recently been improved for use in log anomaly detection [26]. Figure 2.2 depicts an innovative log-based anomaly detection approach that integrates efficient neighbor selection and automated 'k' value determination, comprising three key components. To begin, it employs the minhash algorithm for clustering similar logs into buckets and constructs a Multi-Vantage Point tree (MVP-tree) for the samples within each bucket. Next, it selects 'k' neighbors from the MVP-tree and assesses their relevance using the Silhouette Coefficient, forming the actual neighbor sample set employed for anomaly detection. Finally, the method computes the average distance between samples in the actual neighbor sample set and the target sample to identify anomalies.

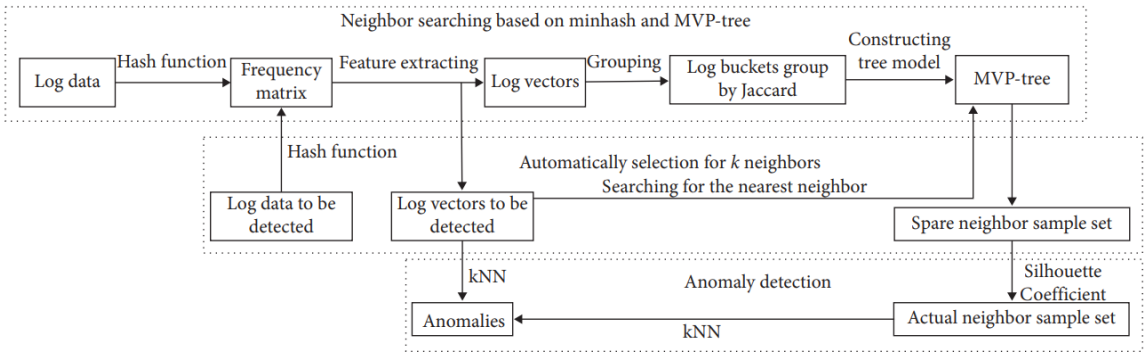

Fig. 2.2: Diagram illustrating the usage of the improved KNN algorithm deployed for log anomaly detection [26].

Support-Vector Machine

SVMs are prominent in academic research and practical applications for their effectiveness in classification and regression tasks. The objective of SVM is to find an optimal hyperplane that maximizes the margin between different classes, with support vectors as key reference points. SVMs are able to handle non-linear data through kernel functions such as polynomial and Radial Basis Function (RBF). A regularization parameter controls

the trade-off between margin maximization and error minimization for when data is not definitively separable.

SVM has also found applications in anomaly detection in textual data, where it is used to classify events from a heterogeneous cybersecurity environment [27]. A method was proposed for detecting security incidents from structured text, in which it leverages the Normalized Compression Distance (NCD) to derive a set of features, which are then used by a SVM.

NCD measure the dissimilarity between a text input T and a set of k additional log entries $\{g_1, g_2, ..., g_k\}$. The result is a vector of k numerical values that represent T within a k-dimensional space. A RBF kernel allowed the SVM to handle the non-linearity of the data. Figure 2.3 illustrates the utilization of SVM in this application. After the datasets are split into two groups, I and G, group G log entries are organized into k generator files, with each file containing logs from a single class. For each log in group I, NCD is utilized to compute the distance between each of the k attribute generators. The matrix is then used as input for the SVM to classify the anomalous data.

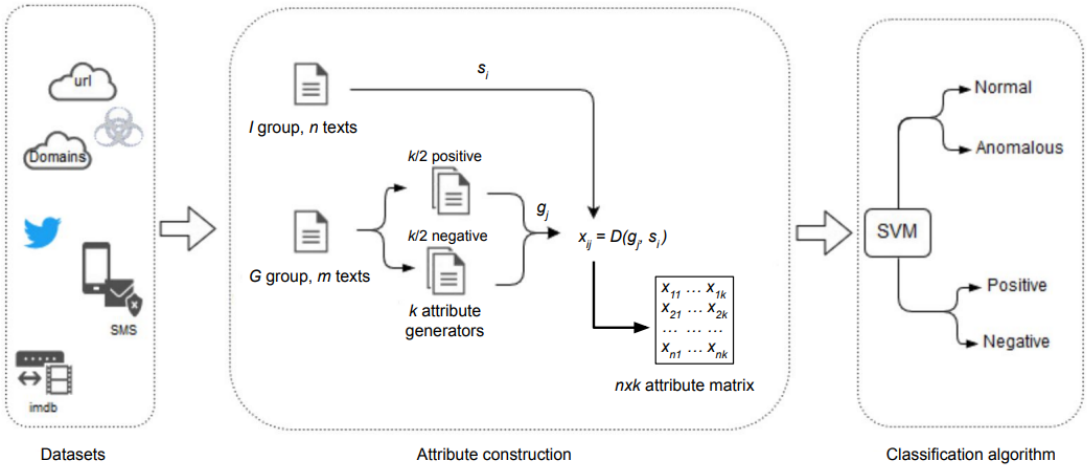

Fig. 2.3: Illustration of the involvement of SVM in a log anomaly detection application [27].

2.1.2 Unsupervised Machine Learning

Unsupervised machine learning operates independently of labeled data, which is essential for scenarios where labeling is limited, such as is often the case with anomaly detection. In the upcoming sections, unsupervised algorithms for anomaly detection will be explored. These include Principal Component Analysis (PCA) and One-Class Support Vector Machine (OCSVM).

Principal Component Analysis

PCA is a statistical technique used for dimension reduction. It involves projecting high-dimensional data into a new coordinate system with a lower dimension (k principal components) while preserving the major characteristics of the original data. PCA identifies the principal components that capture the most variance in the high-dimensional data. It generates two subspaces: the normal space, S_n, composed of the first k principal components and the anomaly space S_a composed of the remaining $n - k$ components, where n is the original dimension.

Figure 2.1 illustrates the usage of PCA on two dimensional data thus to reduce the feature space from two dimensions to one, PCA has selected one dimension to correspond as the normal space, S_n, and the other to be the anomaly space, S_a. As evident by the picture, S_n best described the plotted data, the reasoning for it's selection to be the principal component. The anomalous point highlighted in the figure, does not coincide with the trend of the data. The projection of the vector (2-dimensional point) onto the abnormal subspace is compared with the projection onto the normal subspace and typically a predefined threshold value determines what is an acceptable projection length. Researchers have introduced a fault detection method that employs hierarchical PCA which utilizes distinctive characteristics of dynamic faults for the identification of faults featuring zero cross points (ZCPs) [28].

One-Class Support Vector Machine

OCSVM is primarily used for anomaly detection. Its main objective is to identify anomalies or outliers within a dataset by learning a decision boundary that encapsulates normal data points while minimizing the number of data points inside this boundary. OCSVM seeks to find a hyperplane, that effectively separates the normal data and can apply kernel functions such as RBF like in tradition SVM to handle non-linear data. Unlike SVMs, OCSVMs require only training data from the normal class, as it treats the origin as belonging to a special second class [29]. Figure 2.4 demonstrates a decision boundary for OCSVM and SVM on the same data for comparison.

In recent research, OCSVM has played a significant part in security auditing, specifically in evaluating user access behavior within the information network boundaries of the State Grid Corporation of China (SGCC) [30]. OCSVM is applied by first extracting feature vectors from user behavior data, then training an OCSVM classifier on audit logs representing normal database operation patterns. The classifier is then employed to detect abnormal behavior, enabling the security audit of database user access actions by identifying deviations from established norms.

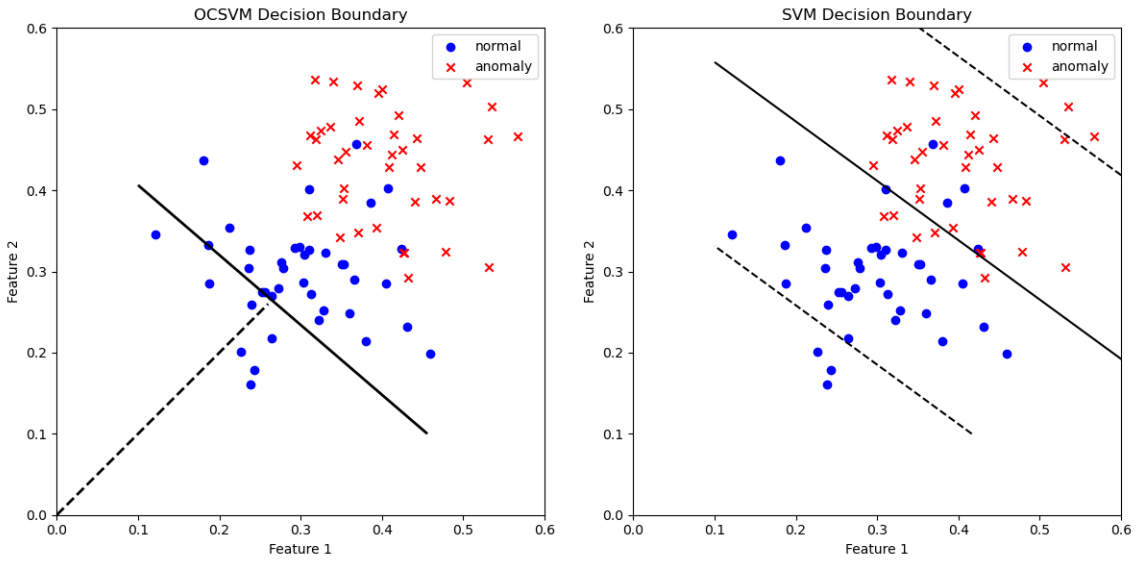

Fig. 2.4: OCSVM (left) and SVM (right) decision boundaries on the same dataset.

LogCluster

LogCluster is an algorithm designed for clustering textual event logs [31]. It identifies frequently occurring words within event logs and extracts these words as tuples for cluster identification. For each event log line, LogCluster creates a tuple of frequent words, enabling it to initialize or update cluster candidates. If a candidate with the same tuple already exists, its support is incremented; otherwise, a new candidate is initialized and its support is set to 1.

A line pattern refers to a structured representation of the common elements in a group of event log lines. These patterns are constructed from the frequent words found in the lines. A line pattern helps capture the characteristics of a cluster of related events.

After generating and updating cluster candidates, LogCluster prunes candidates with a support counter smaller than the user-defined threshold. The remaining candidates, along with their respective line patterns and support levels, are reported as clusters.

This approach is primarily used for clustering textual event logs and offers utility in log anomaly detection. The clustering process involves generating clusters from normal training log data to encapsulate typical log behaviors. Using similarity measures such as distance, anomalies are detected by comparing incoming log entries to established clusters of normal log patterns. LogCluster employs a threshold to categorize log entries as anomalies, classifying entries falling below this threshold as anomalies. A new log sequence is classified as normal or abnormal based on its proximity to clusters.

Isolation Forest

Isolation Forest is an algorithm often used for detecting anomalies. It leverages the principle that anomalies are easier to isolate due to their uniqueness. The algorithm accomplishes this by creating an ensemble of decision trees.

The process of detecting anomalies with Isolation Forest involves partitioning the data using decision trees. Each node in a tree randomly selects a feature and a threshold value within the feature's range. Data points are then divided into two subsets: those below and

above the threshold. The partitioning recursively continues until data points are isolated or a depth limit is reached.

Isolation Forest differentiates anomalies from normal data by calculating the average path length for each data point across all the trees. Shorter path lengths represent fewer required partitions, indicating anomalies. These data points receive higher anomaly scores as they were easily isolated. Data points with longer path lengths align more closely with normal data. The final anomaly score is often derived by aggregating the scores from all trees. A hyperparameter can be set as a threshold to classify data points as anomalies or normal.

2.2 Deep Learning Approaches

In the field of machine learning, the emergence of deep neural networks has brought about a transformative shift in the domain of anomaly detection. Deep learning, a subset of machine learning, harnesses the potential of hierarchical data representation through the layers of neural networks, delivering notably effective results. Contemporary anomaly detection methods based on deep learning have garnered widespread recognition and exhibited superior performance across diverse application domains, surpassing conventional techniques. In the realm of log anomaly detection, models rooted in Recurrent Neural Networks (RNNs), utilizing Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) architectures, such as DeepLog [32], have demonstrated proficiency. Furthermore, more recent advancements in transformer-based log detection models, exemplified by LogBERT [33], have showcased their ability to rival the effectiveness of RNN-based models.

2.2.1 DeepLog

RNN-based models have gained traction in log anomaly detection due to their ability to capture sequential dependencies in log data. RNNs maintain hidden states that capture information from previous entries, allowing them to model temporal dependencies. In particular, RNN-based architectures such as LSTMs and GRUs are preferred for their ability to handle long-range dependencies.

DeepLog leverages the RNN-based model for log anomaly detection. Specifically, DeepLog adopts LSTM to analyze the structure and content of log data [32].

Long Short-Term Memory

The LSTM architecture plays a critical role in DeepLog, enabling the model to effectively capture sequential dependencies inherent in log data. LSTM is a specialized RNN architecture known for its proficiency in modeling sequential data, including log entries. Unlike conventional RNNs, LSTMs are designed to mitigate the vanishing gradient problem, allowing them to capture and retain long-range dependencies intrinsic to sequential data [34].

LSTMs rely on a critical element called the cell state, denoted as C_t, which enables the flow of information across the sequential data with minimal changes. To control the information flow, LSTMs use specialized gates. Figure 2.5 illustrates the LSTM structure for a cell block.

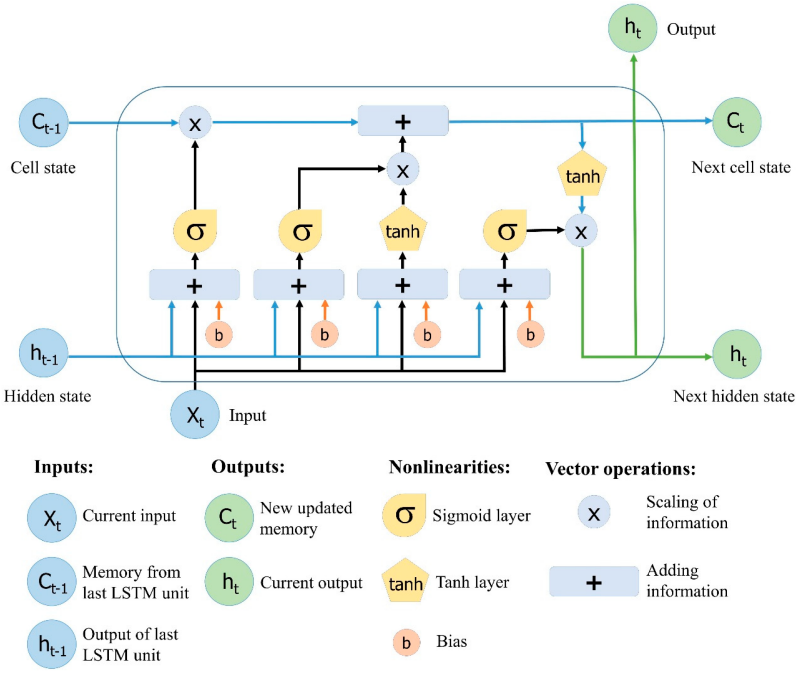

Fig. 2.5: LSTM cell block architecture [35].

The process of updating the cell state in an LSTM is a multi-step procedure that involves managing what information is forgotten from the previous cell state and what new information is added.

Forget Gate (f_t): The forget gate determines what information from the previous cell state (C_{t-1}) should be retained and what should be discarded. It uses a sigmoid activation function to produce values between 0 and 1.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

In this expression:

- $\sigma(z) = \frac{1}{1+e^{-z}}$ is an activation function that maps a real-valued number to a value in the range of 0 to 1, illustrated in Figure 2.6. e represents the base of the natural logarithm.
- W_f is a weight matrix specific to the forget gate. It is used to transform the concatenated input, which consists of two parts: h_{t-1} and x_t.
- b_f: This is the bias term for the forget gate. It is added to the weighted sum before applying the sigmoid activation function.
- $[h_{t-1}, x_t]$ is the concatenation of two vectors: h_{t-1} and x_t, where h_{t-1} indicates the hidden state of the LSTM at the previous time step, and x_t represents the input data at the current time step.

Input Gate (i_t): The input gate controls what new information should be added to the cell state. Like the forget gate, it also uses a sigmoid activation function to regulate the flow of information.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i),$$

where W_i and b_i are learned parameters of the input gate.

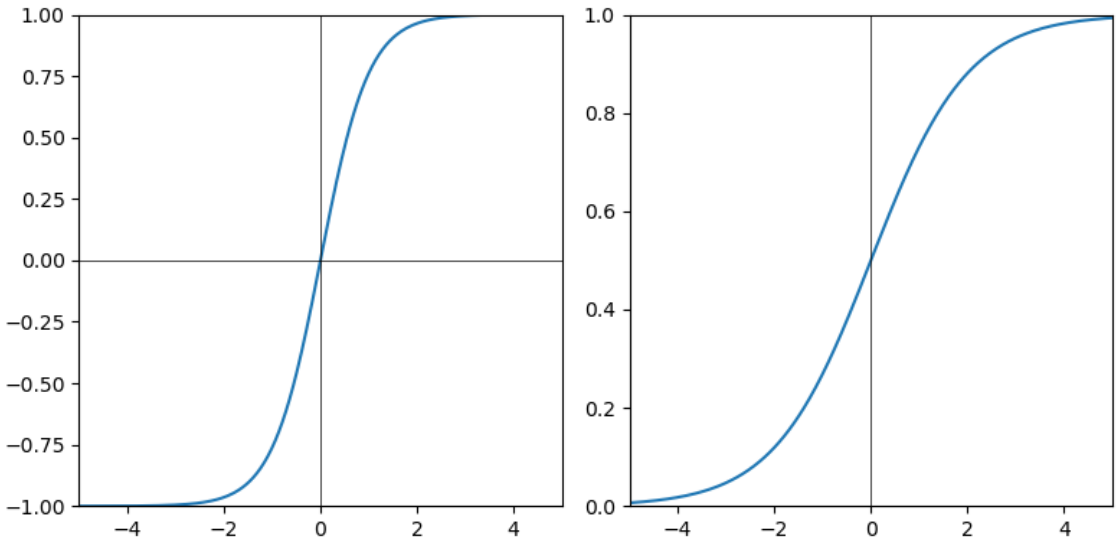

Fig. 2.6: Graphs outlining the hyperbolic tangent function (left) and sigmoid function (right).

Output Gate (o_t): The output gate determines what information should be passed to the next hidden state (h_t) and, ultimately, to the output of the LSTM. It utilizes a sigmoid activation function to decide what parts of the cell state should be included in the output.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o),$$

where W_o and b_o are learned parameters of the output gate.

New Candidate Values ($\tilde{C}_t$): A candidate cell state ($\tilde{C}_t$) is calculated using the hyperbolic tangent (tanh) activation function. This represents the new information that could be added to the cell state.

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c),$$

where tanh is defined as $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, which maps any real-valued number to a value in the range of -1 to 1, illustrated in Figure 2.6; W_c and b_c are learned parameters to calculate the new candidate values.

The cell state (C_t) is updated using the following formula:

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

Here, f_t represents the forget gate's output, i_t is the input gate's output, and $\tilde{C}_t$ is the new candidate value. This equation determines how information changes in the cell state. The forget gate decides what to remove from the previous cell state, and the input gate determines what to add from the new candidate values.

The hidden state (h_t) is computed from the updated cell state as follows:

$$h_t = o_t \cdot \tanh(C_t)$$

This equation utilizes the output gate (o_t), which decides what part of the cell state (C_t) should be exposed as the hidden state. The tanh function is applied to scale the values in the cell state to be within the range of -1 to 1.

Log Anomaly Detection using DeepLog

DeepLog learns patterns from normal log sequences and can identify anomalies by detecting deviations in the log keys or well as their parameter values.

During the preprocessing stage, logs are separated into their respective log keys and parameter value vectors. Two distinct pathways for anomaly detection are employed, one for log keys and the other for parameter values. These pathways combine to create a robust detection method. Figure 2.7 displays the architecture of DeepLog.

For log key sequences, an LSTM component, named the log key anomaly detection model is employed for training. The model learns to predict the next log key in the sequence. In the detection phase, the predicted log keys are critical for determining a sequence to be anomalous. DeepLog incorporates a hyperparameter, denoted as k, which can be fine-tuned. The top k predictions refers to the k log keys with the highest probabilities. Log keys are considered normal if they are in the top k log keys, while anything falling outside of the top

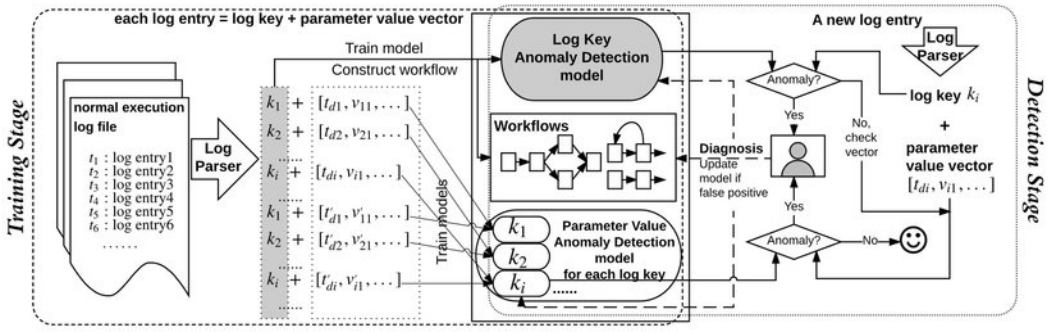

Fig. 2.7: DeepLog Architecture [32].

k predictions is labeled as an anomaly.

Parameter Value Vectors associated with a specific log key are concatenated and treated as a multivariate time-series problem. These series are processed by a separate LSTM model, named the Parameter Value Anomaly Detection model, during training. MSE is used to minimize the loss between the predicted values and the true values. In the detection phase, if a parameter value vector falls within the high confidence interval given by the Gaussian Distribution, it is considered normal.

DeepLog will predict a sequence to be anomalous if either the paramater value anomaly detection model or the log key anomaly detection model predicts an anomaly. It is noteworthy that DeepLog offers the advantage of interpretability by constructing a workflow logic representation that is updated if the model predicts a false positive.

2.2.2 LogBERT

LogBERT is a specialized Transformer-based model designed for log anomaly detection, built upon the Bidirectional Encoder Representations from Transformers (BERT) framework. Figure 2.8 presents an illustrative overview of the LogBERT architecture.

Central to the LogBERT architecture is a Transformer encoder, to provide log sequence processing. This encoder equips LogBERT with the capacity to capture contextual relationships among log keys within sequences, effectively modeling the intricacies of sequential log data. LogBERT's algorithm revolves around two primary objectives: Masked Log Key Prediction (MLKP) and Volume Hypersphere Minimization (VHM).

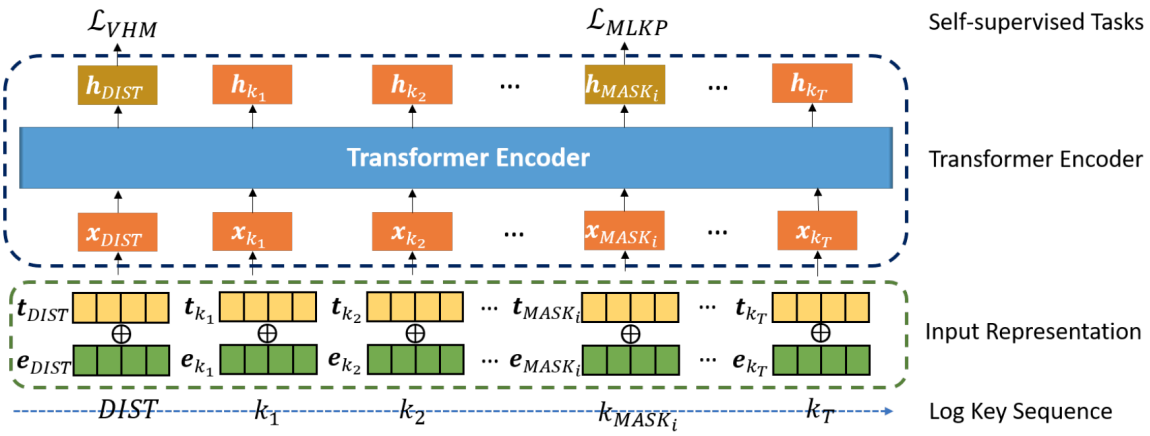

Fig. 2.8: LogBERT Architecture [33].

MLKP entails training LogBERT to predict masked log keys, log keys intentionally obscured during training. This task leverages the known log keys to predict their masked counterparts, equipping LogBERT with an understanding of normal log sequence patterns. This knowledge serves as the foundation for distinguishing normal from anomalous sequences.

VHM introduces an additional dimension to LogBERT's performance. It conceptualizes log key sequence representations within a hypersphere, with the hypothesis that normal log sequences cluster closer to the hypersphere's center, while anomalous sequences occupy more distant regions. Training LogBERT involves minimizing the volume enclosed by this hypersphere. This approach not only enhances anomaly detection but also allows LogBERT to leverage information shared among normal log sequences via the central representation, which encapsulates collective insights from normal sequences. The collaboration between MLKP and VHM tasks empowers LogBERT with robust log anomaly detection capabilities.

CHAPTER 3

Preliminary

3.1 Log Preprocessing

This section discusses log preprocessing, which marks the initial phase of log data analysis. During this phase, raw log entries are organized and converted into more accessible formats, such as vectors. A raw log message is composed of two distinct parts: a constant component and a variable component [36]. Table 3.1 illustrates an instance of log representation, showcasing the separation of logs into templates and lists of values. Templates correspond to the constant component, while the lists of values correspond to the variable component within a raw log message. In a computer system, various templates may exist, each associated with different sets of values.

Table 3.1: Logs, Templates, Values

Log	"Error 574 occurred during step 12794"
Template	"Error *** occurred during step ***"
Values	574, 12794

Unique templates can be given their own id, then a sequence of logs can be represented as a sequence of log ids, making them compatible for models that can handle sequential data such as Recurrent Neural Networks (RNNs).

3.1.1 Log Parsing

The goal of log parsing is to separate the constant part and variable part of a raw log message. The constant component can be referred to as a log event, log key, or log template. The variable component is referred to as a list of values, or parameter values. There are various approaches set to achieve the goal of log parsing: Frequent Pattern Mining,

Clustering, Log-structure Heuristics, Longest Common Subsequence and Evolutionary. [37, 38]. Figure 3.2 presents a visual representation of various log parsing algorithms and their associated characteristics.

Table 3.2: Log Parsing Tools [37].

Log Parser	Year	Technique	Mode	Efficiency	Coverage	Preprocessing	Open Source	Industrial Use
SLCT	2003	Frequent pattern mining	Offline	High	✗	✗	✓	✗
AEL	2008	Heuristics	Offline	High	✓	✓	✗	✓
IPLoM	2012	Iterative partitioning	Offline	High	✓	✗	✗	✗
LKE	2009	Clustering	Offline	Low	✓	✓	✗	✓
LFA	2010	Frequent pattern mining	Offline	High	✓	✗	✗	✗
LogSig	2011	Clustering	Offline	Medium	✓	✗	✗	✗
SHISO	2013	Clustering	Online	High	✓	✗	✗	✗
LogCluster	2015	Frequent pattern mining	Offline	High	✗	✗	✓	✓
LenMa	2016	Clustering	Online	Medium	✓	✗	✓	✗
LogMine	2016	Clustering	Offline	Medium	✓	✓	✓	✗
Spell	2016	Longest Common Subsequence	Online	High	✓	✗	✗	✗
Drain	2017	Parsing tree	Online	High	✓	✓	✓	✗
MoLFI	2018	Evolutionary algorithms	Offline	Low	✓	✓	✓	✗

Log parsing algorithms can be categorized into two types, offline and online. Online algorithms such as Drain are able to parse logs in real-time making them crucial for monitoring and responding to events. Drain belongs to the category of log parsing algorithms centered around the concept of parsing trees. Drain utilizes a fixed-depth parse tree that encodes specifically crafted rules for parsing log data [39].

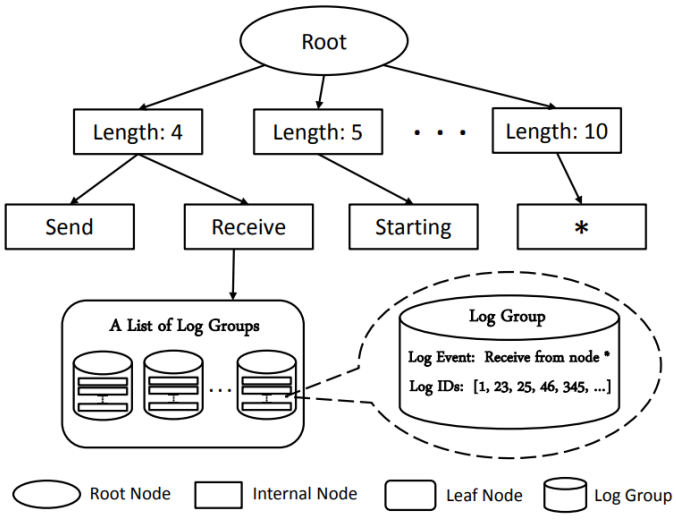

Fig. 3.1: Drain Parse Tree Structure [39].

To streamline log message handling, Drain employs a parse tree with fixed-depth nodes (root, internal, and leaf). This tree guides log group identification, avoiding the need to compare each message with all log groups individually. Leaf nodes store log groups, and a parameter, *maxChild*, helps control the number of nodes visited during the search, improving efficiency. Figure 3.1 illustrates a parse tree structure of depth 3. The root node points to internal nodes that conditionally partition logs. The leaf node contains a list of log groups.

3.2 Deep SVDD

Deep SVDD is a method for one-class classification that employs the concept of a hypersphere to achieve this task [40]. The hypersphere is described by center c and radius R. The center, c, can be calculated from the hidden representation of the sequences as follows:

$$c = \frac{1}{N}\sum_{i=0}^{N} h_i$$

where N is the total number of sequences, and h_i is the hidden representation for the ith sequence. R is determined by the distance between the furthest hidden representation and c.

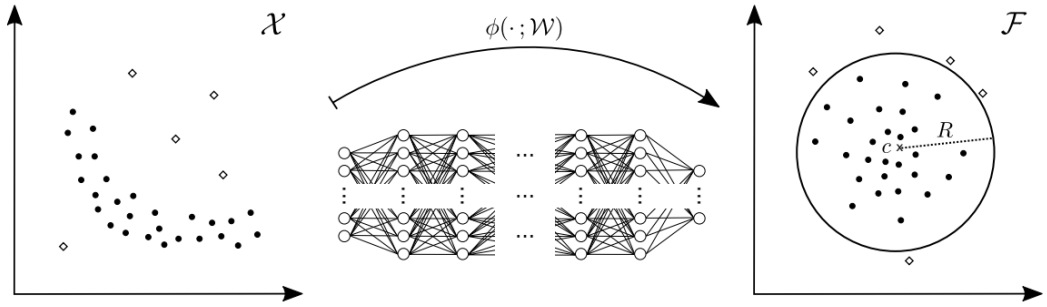

Fig. 3.2: Deep SVDD Transformation [40].

The idea of Deep SVDD is that the mappings of normal samples from the dataset will fall within the hypersphere, while anomalies will map outside of it. Figure 3.2 provides an illustration of the Deep SVDD transformation, showing how data points are mapped into

and outside of the hypersphere.

The Deep SVDD objective function optimizes the parameters of the neural network, $g(\cdot; W)$, to fit a hypersphere. Here, $W = \{w^1, ..., w^L\}$ represents the set of weights for a neural network with L hidden layers. The Deep SVDD objective function is as follows:

$$\min_{W} \frac{1}{N} \sum_{i=1}^{N} ||g(x_i; W) - c||^2 + \frac{\alpha}{2} \sum_{l=1}^{L} ||w^l||^2$$

Breaking down the objective function:

The first term, $\frac{1}{N} \sum_{i=1}^{N} ||g(x_i; W) - c||^2$, measures the average squared distance between the transformed instances, $g(x_i; W)$, from the input X and the hypersphere center, c. This term encourages the model to learn transformations that map normal data points closer to the center of the hypersphere, facilitating the separation of normal and anomalous data. The second term, $\frac{\alpha}{2} \sum_{l=1}^{L} ||w^l||^2$, represents a regularization term that encourages the network to have smaller weights. The hyperparameter α controls the strength of regularization.

3.3 Adversarially Reweighted Learning (ARL)

Adversarial Reweighted Learning (ARL) is a machine learning approach designed to achieve 'fairness' in predictive models. ARL aims to make machine learning models more equitable in their predictions, particularly for underrepresented data points [19]. This approach addresses the need for fairness in machine learning by identifying regions within the data where the model tends to make significant errors and incorporating an adversarial component into the training process to achieve this objective.

ARL works by introducing a minimax game between two key components: the learner, g_θ, and the adversary, λ_ϕ. These components are trained alternately, and their interplay forms the core of ARL's objective function:

$$J(\theta, \phi) = \min_{\theta} \max_{\phi} \sum_{i=1}^{N} \lambda_\phi(x_i, y_i) \cdot \mathcal{L}_{ce}(g_\theta(x_i), y_i)$$

Breaking down this objective function:

The set of parameters, θ, are learned to minimize the expected loss, which is computed using the cross-entropy loss function, denoted as $\mathcal{L}_{ce}$, on the predictions $g_\theta(x_i)$ for input-label pairs (x_i, y_i). For binary classification, the cross-entropy loss is given by:

$$\mathcal{L}_{ce}(g_\theta(x), y) = -[y \log(g_\theta(x)) + (1 - y) \log(1 - g_\theta(x))]$$

In the case of multiclass classification with K classes, it is expressed as:

$$\mathcal{L}_{ce}(g_\theta(x), y) = -\sum_{k=1}^{K} y_k \log(g_\theta(x)_k)$$

The parameters ϕ are learned to maximize the expected loss. Here, λ_ϕ represents an adversarial assignment of weights, calculated as:

$$\lambda_\phi(x_i, y_i) = 1 + N \cdot \frac{f_\phi(x_i, y_i)}{\sum_{i=1}^{N} f_\phi(x_i, y_i)}$$

where f_ϕ is a neural network component of the adversary.

The adversarial component, represented by f_ϕ, plays a crucial role in ARL. It acts as a neural network tasked with identifying regions within the data where the learner (g_θ) makes significant errors. Essentially, it pinpoints areas of the dataset where the model needs improvement, allowing ARL to focus its efforts on these specific regions.

The purpose of the adversarial assignment of weights, λ_ϕ, is to determine how much emphasis is placed on different data points during training. By strategically assigning weights, ARL encourages the learner to improve its performance in regions where errors are most prevalent, ultimately leading to a fairer predictive model.

CHAPTER 4

AdvSVDD

4.1 AdvSVDD Architecture

This section introduces the AdvSVDD framework consisting of two submodel components and the objective function. AdvSVDD is a deep learning neural network for log sequence anomaly detection, based on Deep SVDD. This framework leverages ARL to enhance the performance of anomaly sequence detection trained using the Deep SVDD method. AdvSVDD consists of two main components: the anomaly detector and the adversary. Figure 4.1 illustrates the architecture of AdvSVDD.

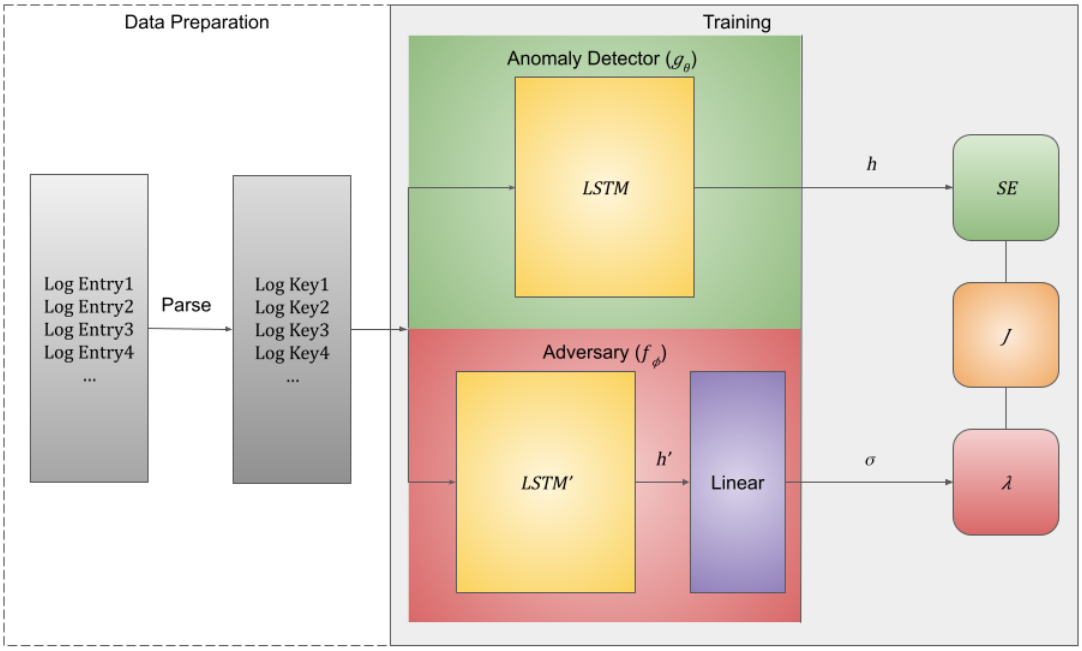

Fig. 4.1: AdvSVDD Architecture.

4.1.1 Anomaly Detector

The anomaly detector, denoted as g_θ, incorporates an LSTM component. The anomaly detector processes input log sequences, x_i, to generate a hidden state, h_i. The final hidden state of the LSTM component for an input sequence can be expressed as $h_i = LSTM(x_i)$. h_i represents the entire sequence in a d-dimensional space and is the output of the anomaly detector, g_θ.

The core of the anomaly detector's training relies on the Deep SVDD loss, encouraging the model to map log sequences within a hypersphere defined by center c. The distance between this center and the sequence with the furthest vector representation corresponds to the radius of the hypersphere. The center and the radius will provide the necessary requirements to build a decision boundary for anomaly detection.

The loss responsible for training the anomaly detector can be defined as the Mean Squared Error (MSE) between the hidden representations, h_i, given by $g_\theta(x_i)$, and the center of the hypersphere, c, which can be formulated as:

$$\frac{1}{N}\sum_{i=1}^{N}||h_i - c||^2$$

This formula encourages the anomaly detector to represent the training data in close proximity to the center and will provide the basis to construct the complete loss function.

4.1.2 Adversary

The adversary submodel in AdvSVDD, similar to the anomaly detector, employs its own LSTM component to represent input log sequences. To discriminate the LSTM component in g_θ and hidden representation it generated, the hidden representation of the sequence generated by the adversary can be written as $h_i^{'} = LSTM'(x_i)$ However, the adversary introduces a crucial distinction: a linear layer. This linear layer maps the hidden state into M dimensions, enabling the adversary to adjust its weights and manipulate the resulting vector values. Subsequently, a sigmoid function is applied to the output vector.

The role of the adversary in AdvSVDD is to enhance the influence of log sequences that are farther from the center. As a result, the anomaly detector interprets sequences with greater distances from the center as anomalies. This functionality proves especially valuable when dealing with smaller training datasets where the hypersphere's surface is not precisely defined based solely on the hidden vector representations.

To express the of transformation of the hidden state h_i' into M dimensions through a linear layer, the subsequent equation can be employed

$$f_\phi(x_i) = \sigma(W_{fr} h_i' + b_f),$$

where $f_\phi(x_i)$ represents the vector produced by the final layer of the adversary model when processing data point x_i. W_{fr} and b_f are the learned matrix and bias parameters. This equation defines how the hidden state h_i' is transformed into M dimensions and subsequently processed using the sigmoid activation function, resulting in the final layer's output for the adversary.

$\lambda_\phi(x_i)$ signifies the adversarial weight assigned to data point x_i by the adversary model. It is computed as the mean of the values produced by the final layer of the adversary model when processing data point x_i. The mathematical expression for $\lambda_\phi(x_i)$ is

$$\lambda_\phi(x_i) = \frac{1}{M} \sum_{j=1}^{M} f_\phi(x_i)_j,$$

where $\lambda_\phi(x_i)$ denotes the adversarial weight for data point x_i and j is an index iterating over the M number of elements of the final layer vector. This equation explicitly defines $\lambda_\phi(x_i)$ as the average of the final layer values produced by the adversary model for data point x_i, quantifying the extent to which the adversary assigns importance to that specific data point during adversarial training.

4.1.3 Objective Function

The essence of the ARL equation is to use λ_ϕ as a weight to the loss calculated for a

particular data point and the target. In the case of AdvSVDD, the data point is the hidden representation of x_i, h_i, and the target is the center of the hypersphere, c. Thus, λ_ϕ must be the weight associated with the squared error (SE) between h_i and c. In its simplest description, the loss for the anomaly detector is a Mean of *Weighted* Squared Errors and can be defined as follows:

$$\frac{1}{N}\sum_{i=1}^{N}\lambda_\phi(x_i)\cdot||h_i-c||^2$$

The loss for the adversary is simply the negative of the anomaly detector loss value. Thus, the completed objective function of AdvSVDD, denoted as $J(\theta,\phi)$, is formulated as follows:

$$J(\theta,\phi)=\min_\theta\max_\phi\frac{1}{N}\sum_{i=1}^{N}\lambda_\phi(x_i)\cdot||h_i-c||^2$$

This equation consists of two sets of parameters, θ and ϕ, that are to be optimized. θ corresponds to the parameters of the anomaly detector, while ϕ corresponds to the parameters of the adversary. The objective function entails two optimization operations: $\min_\theta$ and $\max_\phi$. The min operation seeks to find values for θ that minimize the function, while the max operation aims to find values for ϕ that maximize the function. The optimization is performed over N log sequences, and the weight $\lambda_\phi(x_i)$ influences the magnitude of the squared error for each sequence, making it a dynamic factor in the overall objective.

4.2 Training and Detection

In this section, the AdvSVDD training framework is outlined, encompassing scheduling parameters, foundational anomaly detector training, and submodel alternation training. The section concludes with a brief overview of anomaly detection with AdvSVDD.

4.2.1 Training Schedule

Specific parameters play a pivotal role in shaping the AdvSVDD training schedule:

1. **Total Number of Epochs (T)**: This parameter dictates the overall duration of AdvSVDD training, representing the total number of epochs.

2. **Initial Anomaly Detector Duration (s)**: The initiation of the submodel alternation training phase occurs at a defined epoch s.

3. **Cycle Duration (u)**: Within the submodel alternation training phase (spanning from epoch s to T), the training is organized into cycles, each covering a fixed number of epochs denoted as u.

The entire training process spans T epochs. Prior to the starting point, s the anomaly detector undergoes foundational training. Epoch s marks the commencement of the submodel alternation training phase. Each cycle lasts for u epochs. The training phases are illustrated in Figure 4.2.

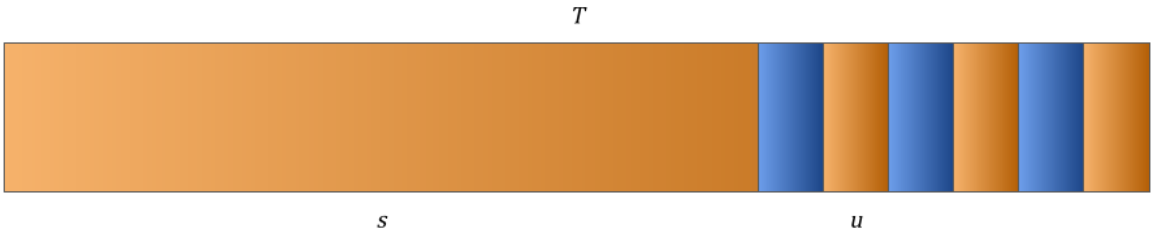

Fig. 4.2: The training progresses from left to right. The orange portion of the figure corresponds to the anomaly detector training. The blue portion of the figure corresponds to the adversary training.

4.2.2 Foundational Anomaly Detector Training

In phase one, the anomaly detector trains independently for a set number of epochs, s, without the presence of an adversary. This separation is critical because, during this initial phase, the model has not yet acquired a comprehensive understanding of the intricacies within the training data. During this period, the adversarial weights are set to be an M dimensional vector consisting of ones. The result is reducing the adversarial weight to 1, effectively simplifying the loss function to the MSE.

4.2.3 Submodel Alternation Training

In phase two, the adversary is introduced into the training process. The output from

f_ϕ is used to calculate λ_ϕ to serve as a unique weight for the SE of h_i and c. This weighted loss is employed to train the adversary, causing the output from f_ϕ to update the final resulting weight, λ_ϕ.

The underlying principle governing this method revolves around cyclic alternation between two crucial submodels: the anomaly detector and the adversary. This alternation occurs at regular intervals of u epochs, ensuring that only one submodel is actively trained at any given time.

4.2.4 Detecting Anomalies with AdvSVDD

Detecting anomalies with AdvSVDD involves a straightforward process. During the training phase, the algorithm minimizes the volume of the hypersphere. This minimization leads to the reduction of the radius (R), which is defined as the distance between the center of the hypersphere and the farthest hidden representation. The obtained center (c) and R together form a decision boundary represented by the surface of the hypersphere. In the evaluation phase, incoming data points (x_i) are classified as normal if their hidden representation (h_i) distance to c is smaller than R (within the hypersphere). Conversely, data points are classified as anomalous if their h_i has a distance to c that exceeds R (outside the hypersphere).

The adversarial weight, (λ_ϕ) plays a crucial role in this process. It is applied to the squared error between a hidden representation and c. As a result, the learned radius used to establish the decision boundary is influenced. Specifically, the model can place emphasis on points closest to the surface of the hypersphere.

CHAPTER 5

Experiments

5.1 Experimental Setup

In this section, the foundational elements of the experimental setup will be presented. Commencing with an introduction to the datasets. Following the dataset overview, the data splitting strategy will be expounded upon. The baseline models and the evaluation metrics will be introduced. Finally, this section will conclude by presenting implementation details of the AdvSVDD model.

5.1.1 Datasets

Two datasets used in this project: the Blue Gene/L (BG/L) Dataset and the Thunderbird Dataset. Both datasets are massive and provide both non-anomalous and anomalous logs.

Blue Gene/L

The BG/L dataset, is a collection of data from the Blue Gene/L supercomputer system deployed at Lawrence Livermore National Labs (LLNL). Developed by International Business Machines Corporation (IBM), Blue Gene/L is designed with high-performance computing capabilities and capacity for massive parallel processing.

For data preprocessing, the Drain log parser was used to extract structured information from raw and unstructured log messages. The column structure of the BG/L dataset includes fields such as LineId, Label, Timestamp (recorded date and time), Node, Time, NodeRepeat, Type, Component, Level, Content, EventId, EventTemplate, and ParameterList. Additionally, a sliding window approach with a window size of 20 and a step size of 10 was applied to the first 5,000,000 rows to split the log files into sequences.

Thunderbird

The Thunderbird dataset is a collection of data from the Thunderbird supercomputing system produced by Dell and installed at Sandia National Labs (SNL). The Thunderbird supercomputing system is a prominent high-performance computing (HPC) system primarily employed for scientific and research purposes. This dataset encompasses various logs and event information generated by the Thunderbird system.

Similar to the BG/L dataset, data preprocessing involved the use of the Drain log parser to extract structured information from raw log messages. The column structure of the Thunderbird dataset includes fields such as LineId, Label, Timestamp (date and time of the log entry), User, Month, Day, Time, Location, Component, PID, Content, EventId, EventTemplate, and ParameterList. The same sliding window approach with a window size of 20 and a step size of 10 was was applied to the first 5,000,000 rows to split log files into sequences.

Dataset Split Details

The splitting technique is identical for both the BG/L and Thunderbird datasets. As the sliding window was applied to the dataset. The entries were filtered to contain two columns: EventId and Sequence_label. EventId is a list of the ids corresponding to the id of each log template. The Sequence_label column contains the label for if the sequence is anomalous or normal.

The dataset is split into three sets: a training set with 10,240 samples, a validation set with 11,264 samples, and a testing set with 11,264 samples. The class distribution within these sets includes 30,720 normal samples and 2,048 abnormal samples. Notably, the training set comprises entirely normal samples, while both the validation and testing sets include 1,024 abnormal samples for a resulting ratio of ten normal samples for every one abnormal sample. The training, validation and test sets are saved into a csv file to be used indefinitely. All experiments were conducted using these csv files and sampled from when specified.

The split resulted in the Thunderbird csv file containing a total of 523 unique log keys, compared to the BG/L csv file containing 114 unique log keys). Each sequence contains a total of 20 log keys.

5.1.2 Baselines

As baseline models used for comparison, the following were implemented and evaluated:

- Isolation Forest (iForest) was implemented using 100 estimators.
- One-Class Support Vector Machine (OCSVM) was implemented, and the validation set was used to find the best-performing nu hyperparameter within the range of 10^{-5} to 10^{-2}, with a step size of 5×10^{-5}.
- The DeepLog component, Log Key Anomaly Detection model, for log sequence anomaly detection was implemented. Using a hidden dimension of 128, two LSTM layers and a learning rate of 10^{-5}. Furthermore, during the validation phase, the hyperparameter k was tuned for best performance.
- Support Vector Data Description (SVDD) was selected to evaluate the adversary component of the proposed AdvSVDD model. When the adversary component of AdvSVDD is turned off for the entirety of the training, it corresponds directly as the SVDD baseline. The hyperparameters are identical between the AdvSVDD and SVDD models.

All models were trained on the exact same data, sampled from the training set using the same seed.

5.1.3 Evaluation Metrics

Precision, Recall, F1-score, ROC AUC (Receiver Operating Characteristic Area Under Curve), and PR AUC (Precision Recall Area Under the Curve) are utilized to assess the models throughout the validation and testing phases. The F1-score for the anomalous class was used as the criteria for selecting the best model during the validation phase.

Precision measures the model's ability to correctly identify positive instances. High precision indicates the model's effectiveness in minimizing false positive errors, which is especially important in applications like log sequence anomaly detection. It is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall, also known as sensitivity, evaluates the model's capacity to detect all relevant positive instances within a dataset. High recall is essential when missing positive cases could have significant consequences, as is often the case with anomaly detection. It is calculated as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

The F1-Score strikes a balance between precision and recall, offering a composite measure of a classifier's performance. This metric is useful for evaluating models on imbalanced datasets, as it considers false positives and false negatives equally. It is calculated as:

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

The ROC AUC metric assesses a model's ability to distinguish between positive and negative classes by examining the area under the Receiver Operating Characteristic curve. A higher ROC AUC value indicates better discrimination capability.

PR AUC is another binary classification metric that considers the trade-off between precision and recall. It is particularly suitable for imbalanced datasets and is calculated by examining the area under the Precision-Recall curve. A higher PR AUC value signifies a model's ability to maintain a high level of precision while preserving an acceptable level of recall.

5.1.4 Implementation Details

The AdvSVDD implementation involves several crucial hyperparameters and configurations. Initially, the batch sizes for the training, validation, and testing datasets are set

at 32, determining the number of data points processed simultaneously during training and evaluation. To ensure impartiality in training and evaluation, all models, including the baseline models, undergo training with the same ten seed values.

The AdvSVDD architecture comprises an anomaly detector with an embedding dimension of 50 and a hidden dimension of 256, incorporating a single LSTM layer. Similarly, the adversary network also employs a single LSTM layer with matching embedding and hidden dimensions. The AdvSVDD was scheduled to begin the submodel alternation training phase at epoch 210, with a cycle duration of 5 epochs.

Adam Optimization

Both the anomaly detector and the adversary are optimized utilizing the Adam algorithm [41]. Adam has proven to be a powerful optimization algorithm by combining the two key concepts of momentum in AdaGrad [42] and the exponentially weighted averages in RMSprop [43].

Adam was utilized with a learning rate of 10^{-2}, with a β_1 (for momentum) and β_2 (for exponentially weighted averages) values of 0.9 and 0.999 respectively.

5.2 Experimental Results

In this section, experimental results are outlined for all evaluation metrics, followed by an examination of the influence of the adversary during training. The investigation explores the effects of varying the number of normal samples for training and employing different training schedules for AdvSVDD. Additionally, the section includes a sensitivity analysis on the number of dimensions in the final linear layer of the adversary.

5.2.1 Overall Performance

Table 5.1 provides an overview of average performance metrics for anomaly detection models on the BG/L and Thunderbird datasets. For each model and dataset, the values are presented with their corresponding standard deviations.

Table 5.1: Average Performance Metrics for Anomaly Detection Models

Dataset	Model	Precision	Recall	F1	ROC AUC	PR AUC
BG/L	iForest	$0.1466_{\pm 0.006}$	$0.3212_{\pm 0.001}$	$0.2013_{\pm 0.006}$	$0.5670_{\pm 0.004}$	$0.2648_{\pm 0.003}$
	OCSVM	$0.6703_{\pm 0.082}$	$0.8982_{\pm 0.204}$	$0.7636_{\pm 0.139}$	$0.9281_{\pm 0.102}$	$0.7889_{\pm 0.133}$
	DeepLog	$0.8052_{\pm 0.191}$	$0.8620_{\pm 0.113}$	$0.8104_{\pm 0.114}$	$0.9025_{\pm 0.060}$	$0.8526_{\pm 0.086}$
	SVDD	$0.8984_{\pm 0.060}$	$0.9955_{\pm 0.009}$	$0.9436_{\pm 0.034}$	$0.9919_{\pm 0.006}$	$0.9472_{\pm 0.031}$
	AdvSVDD *	$0.9144_{\pm 0.054}$	$0.9942_{\pm 0.011}$	$0.9519_{\pm 0.031}$	$0.9923_{\pm 0.006}$	$0.9546_{\pm 0.028}$
Thunderbird	iForest	$0.2697_{\pm 0.006}$	$0.8957_{\pm 0.009}$	$0.4145_{\pm 0.007}$	$0.8265_{\pm 0.004}$	$0.5874_{\pm 0.004}$
	OCSVM	$0.5173_{\pm 0.016}$	$0.9896_{\pm 0.010}$	$0.6793_{\pm 0.014}$	$0.9486_{\pm 0.005}$	$0.7539_{\pm 0.009}$
	DeepLog	$0.3552_{\pm 0.278}$	$0.4380_{\pm 0.182}$	$0.2899_{\pm 0.041}$	$0.6345_{\pm 0.039}$	$0.4221_{\pm 0.079}$
	SVDD	$0.6968_{\pm 0.184}$	$0.9214_{\pm 0.068}$	$0.7823_{\pm 0.144}$	$0.9356_{\pm 0.049}$	$0.8127_{\pm 0.110}$
	AdvSVDD *	$0.7086_{\pm 0.149}$	$0.9218_{\pm 0.094}$	$0.7949_{\pm 0.117}$	$0.9397_{\pm 0.054}$	$0.8187_{\pm 0.101}$

Precision

In both the BG/L and Thunderbird datasets, AdvSVDD outperforms other models in terms of precision. It achieves a precision of $0.9144_{\pm 0.054}$ in the BG/L dataset and $0.7086_{\pm 0.149}$ in the Thunderbird dataset. On the other hand, iForest consistently exhibits the lowest precision in both datasets, with scores of $0.1466_{\pm 0.006}$ in BG/L and $0.2697_{\pm 0.006}$ in Thunderbird.

In the BG/L dataset, OCSVM and DeepLog perform better than iForest in precision, with scores of $0.6703_{\pm 0.082}$ and $0.8052_{\pm 0.191}$, respectively. SVDD achieves a precision of $0.8984_{\pm 0.060}$. AdvSVDD leads in the BG/L precision scores with $0.9144_{\pm 0.054}$, a 1.6% difference with second place.

In the Thunderbird dataset, AdvSVDD $0.7086_{\pm 0.149}$ remains the top performer in precision. With SVDD at a close second with a score of $0.6968_{\pm 0.184}$.

Recall

In both the BG/L and Thunderbird datasets, AdvSVDD demonstrates strong recall performance in anomaly detection. Surprisingly, iForest exhibits a notably high recall in the Thunderbird dataset, with a score of $0.8957_{\pm 0.009}$ despite a low recall of $0.3212_{\pm 0.001}$ in the BG/L dataset. This is opposite to DeepLog, which initially exhibited a higher recall in the BG/L dataset. DeepLog has a significant decline in recall when applied to the Thunderbird dataset, transitioning from a recall of $0.8620_{\pm 0.113}$ to a recall score of $0.4380_{\pm 0.182}$.

In the BG/L dataset, AdvSVDD achieves a recall of $0.9942_{\pm 0.011}$, positioning it among the top performers in correctly identifying anomalies. However, it's worth noting that AdvSVDD is the second-highest performer in recall, slightly trailing SVDD, which attains a recall of $0.9955_{\pm 0.009}$.

In the Thunderbird dataset, AdvSVDD has a slight edge over SVDD in terms of recall, with AdvSVDD achieving a recall of $0.9218_{\pm 0.094}$ compared to SVDD's recall of $0.9214_{\pm 0.068}$. It's noteworthy that OCSVM exhibits the highest recall in this dataset, surpassing both AdvSVDD and SVDD with a score of $0.9896_{\pm 0.010}$.

F1-Score

AdvSVDD demonstrated superior performance in the F1-Score metric, outperforming all other models on both datasets. Specifically, AdvSVDD achieved a score of $0.9519_{\pm 0.031}$ in the BG/L dataset and 0.7949 ± 0.117 in the Thunderbird dataset. Notably, across both datasets, AdvSVDD not only surpassed SVDD in F1-Score but also showcased a lower standard deviation.

An intriguing observation emerges as all the deep learning models exhibit a notable decline in performance when transitioning from the BG/L dataset to the Thunderbird dataset. In contrast, iForest experiences an increase in performance. Specifically, in the BG/L dataset, iForest achieved an F1-Score of $0.2013_{\pm 0.006}$. However, in the Thunderbird dataset, iForest's performance notably improves, yielding a score of $0.4145_{\pm 0.007}$.

PR AUC and ROC AUC

In both the BG/L and Thunderbird datasets, the comparisons of the models through the lens of the PR-AUC metric are essentially the same as F1-Score, with different values. Through the lens of the ROC AUC metric, the comparisons are the same in the Thunderbird dataset, and slightly different in the BG/L dataset. In the BG/L dataset, the OCSVM achieves a ROC AUC score of $0.9281_{\pm 0.102}$ overtaking DeepLog's ROC AUC score of $0.9025_{\pm 0.060}$.

Standard Deviations

An intriguing observation arises regarding the variability in precision and recall for AdvSVDD compared to SVDD. Across both datasets, AdvSVDD exhibited a lower standard deviation than SVDD in precision (BG/L: 0.054 vs. 0.060, Thunderbird: 0.149 vs. 0.184), but its standard deviation in recall was larger (BG/L: 0.011 vs. 0.009, Thunderbird: 0.094 vs. 0.068). However, it is noteworthy that the increase in standard deviation in recall was comparatively smaller than the decrease in the standard deviation in precision. This results in AdvSVDD having a smaller standard deviation for F1-Score than SVDD (BG/L: 0.031 vs. 0.034, Thunderbird: 0.117 vs. 0.144).

Examining the F1-Score and PR AUC metrics reveals that when seeking a balance between Precision and Recall, AdvSVDD appears to be more dependable than SVDD. It consistently maintains a lower standard deviation across both datasets while delivering superior performance.

Table 5.2: Training Runtime Comparison

Dataset	**Model**	**Time (Min:Sec)**
BG/L	DeepLog	49:14
	SVDD	5:50
	AdvSVDD	5:58
Thunderbird	DeepLog	217:57
	SVDD	5:49
	AdvSVDD	5:52

Training Runtime Comparison

Table 5.2 displays the runtime of different models on the BG/L and Thunderbird datasets. Runtimes are presented for DeepLog, SVDD, and AdvSVDD models.

There are two key takeaways from the runtime comparison.

1. AdvSVDD and SVDD benefit from the fact that their optimization does not depend on the number of log keys. For DeepLog, the runtime scaled with the number of

unique log keys in the datasets (BG/L: 114, Thunderbird, 523) due to the validation phase optimizing the k hyper-parameter as explained in 5.1.2.

2. The addition of the Adversary in AdvSVDD only results in a minimal increase in runtime compared to SVDD. This makes sense since for any given epoch in phase two, either the Anomaly Detector or Adversary is being trained. The adversary has an additional linear layer for training.

5.2.2 Investigating AdvSVDD

Experiments were conducted to investigate the impact of adversary during training. The studies were designed for the purpose of evaluating AdvSVDD performance under different conditions: the influence of different training schedules on AdvSVDD and the effect of varying the number of normal samples used for training.

Scheduling Variants for Training AdvSVDD

The study evaluated three scheduling variants for AdvSVDD: AdvSVDD-1, AdvSVDD-2, and AdvSVDD-3. All variants were assessed on the BG/L and Thunderbird datasets, each trained on 1024 normal training samples. All variants utilized a training schedule with $T = 300$. Thus, trained for a total of 300 epochs.

s represents the number of epochs allocated to training the anomaly detector before involving the adversary. For AdvSVDD-1, $s = 205$, and the training schedule involved alternating cycle durations. The first, (u_1), lasted for 10 epochs, focusing on training the anomaly detector, followed by a cycle (u_2) with a duration of 5 epochs, dedicated to adversary training. This cycling continued until T was reached.

AdvSVDD-2 and AdvSVDD-3 used a similar approach with cycle durations, but simplified it by combining u_1 and u_2 into a single value, u. The cycle duration was set to $u = 5$ for both variants. AdvSVDD-2 had a schedule with $s = 220$, while AdvSVDD-3 had a schedule with $s = 210$.

In Table 5.3, the average performance metrics for the three variants are presented.

Table 5.3: Average Performance of AdvSVDD with Different Training Schedules

Dataset	Model	Precision	Recall	F1 Score	ROC AUC	PR AUC
BG/L	SVDD	$0.8984_{\pm 0.060}$	$0.9955_{\pm 0.009}$	$0.9436_{\pm 0.034}$	$0.9919_{\pm 0.006}$	$0.9472_{\pm 0.031}$
	AdvSVDD-1	$0.9047_{\pm 0.067}$	$0.9949_{\pm 0.011}$	$0.9464_{\pm 0.037}$	$0.9919_{\pm 0.007}$	$0.95_{\pm 0.033}$
	AdvSVDD-2	$0.9091_{\pm 0.061}$	$0.9941_{\pm 0.012}$	$0.9487_{\pm 0.034}$	$0.9918_{\pm 0.007}$	$0.9518_{\pm 0.031}$
	AdvSVDD-3 *	$0.9144_{\pm 0.054}$	$0.9942_{\pm 0.011}$	$0.9519_{\pm 0.031}$	$0.9923_{\pm 0.006}$	$0.9546_{\pm 0.028}$
Thunderbird	SVDD	$0.6968_{\pm 0.184}$	$0.9214_{\pm 0.068}$	$0.7823_{\pm 0.144}$	$0.9356_{\pm 0.049}$	$0.8127_{\pm 0.110}$
	AdvSVDD-1 *	$0.7179_{\pm 0.15}$	$0.922_{\pm 0.083}$	$0.8017_{\pm 0.118}$	$0.9408_{\pm 0.05}$	$0.8235_{\pm 0.102}$
	AdvSVDD-2	$0.7057_{\pm 0.159}$	$0.9367_{\pm 0.085}$	$0.7996_{\pm 0.133}$	$0.9462_{\pm 0.055}$	$0.8241_{\pm 0.112}$
	AdvSVDD-3	$0.7086_{\pm 0.149}$	$0.9218_{\pm 0.094}$	$0.7949_{\pm 0.117}$	$0.9397_{\pm 0.054}$	$0.8187_{\pm 0.101}$

An interesting observation is that AdvSVDD-3 excelled in the BG/L dataset while AdvSVDD-1 exhibited the poorest performance. However, on the Thunderbird dataset, AdvSVDD-1 outperformed the other variants, with AdvSVDD-3 delivering the weakest performance. The scores for SVDD are also in the table for comparison.

AdvSVDD-3 consistently performs the best among the three variants. This could indicate that a longer initial training phase, larger (s), for the anomaly detector is beneficial for the BG/L dataset.

On the Thunderbird dataset, AdvSVDD-1 emerges as the top performer, especially in terms of Precision and F1 Score. This implies that, for the Thunderbird dataset, a shorter initial training phase (s) for the anomaly detector yields better results. Notably, the choice of cycle duration appears to play an important role in performance, as this is the only variant that allows the anomaly detector to train for twice as long as the adversary during the cyclic training phase.

One of the most significant findings is the dataset-specific performance of the variants, highlighting the need to adapt the training schedule to suit the specific characteristics of the dataset, as there is no single solution for all datasets. The variations in dataset-specific performance among the AdvSVDD variants underscore the importance of tailoring the training schedule and cycle duration to the dataset.

Figure 5.1 Demonstrates the performance on both datasets for the three variant schedules, including no schedule (SVDD) throughout the ten different seed values.

Beginning with the BG/L dataset, SVDD struggled with seed 1 whilst all AdvSVDD

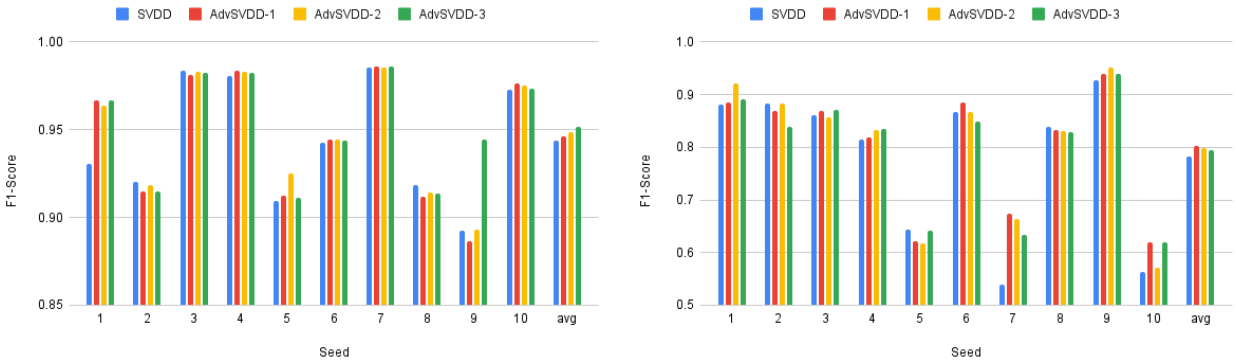

Fig. 5.1: Bar charts illustrating the F1-scores of variant schedules on the BG/L (left) and Thunderbird (Right) datasets.

variants outperformed. SVDD slightly edges in some seeds but it's always minimal when compared to the runs where the AdvSVDD variants show superior performance. The largest performance boost happened in seed 9 with AdvSVDD-3 outperforming the rest exhibiting an F1-score of 0.9442 whilst the rest of the models perform under 0.90.

The results for the Thunderbird dataset shows a similar phenomena. The SVDD model slightly edges in some seeds, such as seed 8, but it is minimal with compared to the performance boosts when the adversary models outperform. Seed 7 demonstrates the most evident example.

Varying the Number of Training Samples

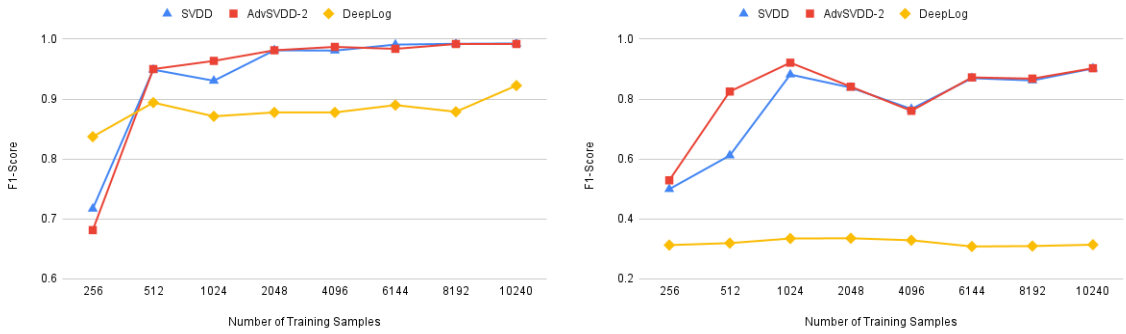

Fig. 5.2: F1-Scores of SVDD, AdvSVDD, and DeepLog with Varying Training Samples

In Figure 5.2, F1-Scores are compared for SVDD, AdvSVDD, and DeepLog models with increasing training samples on the BG/L (left) and Thunderbird (right) datasets. Significantly, SVDD and AdvSVDD consistently outperform DeepLog on both datasets. As training samples increase, AdvSVDD and SVDD show similar performance trends, with more pronounced differences in datasets below 2048 samples. The Thunderbird dataset, with a significantly higher number of unique log keys (523) compared to BG/L (114 unique log keys), highlights these distinctions. The disparity in log keys potentially contributes to diminished performance on the Thunderbird dataset.

An interesting trend emerges: AdvSVDD tends to converge toward SVDD performance as training samples increase. This aligns with expectations, where a larger dataset better represents the hypersphere. Notably, the efficacy of adversarial weight-induced variability shines in scenarios with smaller training samples, enhancing the model's ability to generalize in the absence of an optimal hypersphere definition.

5.2.3 Sensitivity Analysis of Final Linear Layer Dimensions

The final linear layer in the adversary was examined by varying the number of dimensions, M. The following dimensions were tested: $1, 8, 16, 32, 64$.

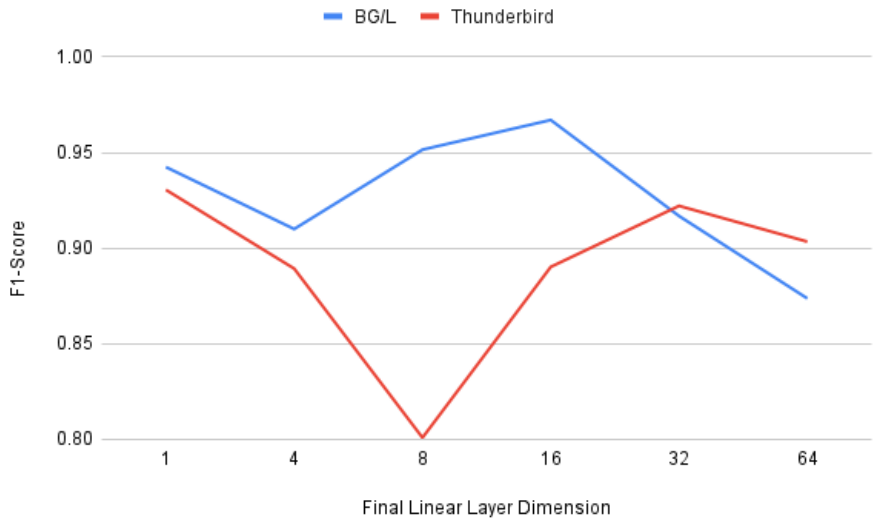

Fig. 5.3: Sensitivity Analysis of Final Linear Layer Dimensions.

Figure 5.3 illustrates AdvSVDD-3's F1-scores on 1024 training samples using different final linear layer dimensions on the BG/L and Thunderbird datasets. The figure suggests that mapping to a single dimension performs relatively well. Following is an initial drop off in performance until the dimensions are tuned. Exceeding the tuning results in another drop in performance. This phenomena is present in both datasets. However, the line depicting BG/L demonstrates it with lower dimension values. This could be potentially be as a result of the differences in the datasets, particularly the amount of log keys.

5.2.4 Visualization

Figure 5.4 (BG/L) and Figure 5.5 (Thunderbird) illustrate hidden representations of normal and anomalous log sequences in the test data. These representations were obtained after training AdvSVDD on 1024 normal training samples. The 256 dimensions of the hidden representations were reduced to two dimensions using PCA.

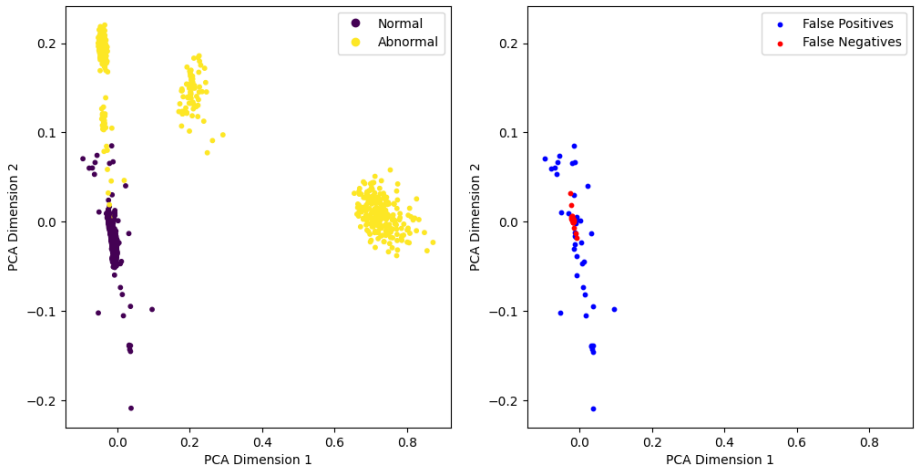

Fig. 5.4: Dimensionality Reduciton Plot for BG/L

The BG/L dataset plot displays distinct clusters of abnormal sequences, while the Thunderbird plot lacks evident anomaly clusters. Notably, false positives (depicted in blue) are distant from the majority of mappings, aligning with expectations of a distance-based decision criterion. As anticipated, false negatives (depicted in red) are infrequent in both

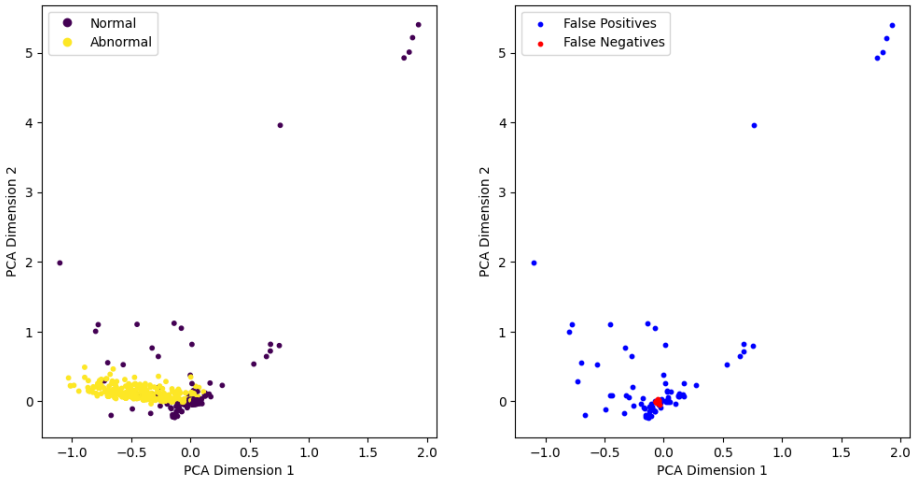

Fig. 5.5: Dimensionality Reduciton Plot for Thunderbird

datasets and occur when anomalous points are closely mapped to the center.

An additional observation focuses on the range spanned by both axes in the BG/L and Thunderbird plots. Particularly, the Thunderbird plot exhibits a larger span in both axes. This suggests that the model encountered difficulties in effectively mapping normal points to the center of the hypersphere.

The variation in mapping effectiveness could be a factor in the distinctive clustering observed in the BG/L dataset, where anomalous points tend to cluster apart from the majority of normal points. Conversely, the plot of the Thunderbird dataset does not depict clear clusters of anomalous data points, suggesting a different mapping dynamic.

CHAPTER 6

Conclusion

The field of log sequence anomaly detection holds significant academic and practical importance, with applications in areas such as cybersecurity, network surveillance, industrial processes, and financial transaction monitoring. This thesis introduces the AdvSVDD architecture, a deep learning model designed to address the challenge of limited training data in sequence anomaly detection.

6.1 Key Findings and Contributions

AdvSVDD excels in identifying anomalies within log sequences and enhancing the sequence anomaly detection process under limited training data through the integration of two key concepts. Deep SVDD, a One-Class Classification framework and Adversarially Reweighted Learning.

AdvSVDD was evaluated on two datasets, BG/L and Thunderbird using percision, recall, F1-Score, ROC AUC and PR AUC. Comparisons with baseline models and a non-adversarial SVDD model demonstrated AdvSVDD's superiority, emphasizing the impact of the adversarial component.

The investigation on the different training schedules demonstrated the enhancement effect of the adversarial component in AdvSVDD as all variations outperformed the traditional SVDD model. The study on the number of normal samples for training highlighted that the impact of the AdvSVDD framework is most effective with limited training data, and the involvement of the adversary does not hinder the performance as the number of training data increases. The sensitivity analysis provided some insight into the dimensional effect on final linear layer of the adversary. Demonstrating that it is dataset-dependent but follows a typical pattern across both the BG/L and Thunderbird dataset.

6.2 Practical Implications and Impact

The integration of adversarial techniques in AdvSVDD offers significant implications for log sequence anomaly detection. Its ability to excel in scenarios with limited training data makes it suitable for data-scarce applications.

The inclusion of the adversarial component within AdvSVDD becomes evident through rigorous comparisons with the traditional SVDD model. The enhanced performance of AdvSVDD underscores the effectiveness of the adversarial training schedule. This has significant implications for researchers seeking to leverage adversarial approaches in their anomaly detection models.

Beyond its practical implications, the AdvSVDD framework serves as a foundational reference for future research in adversarial anomaly detection. It has the potential to stimulate further exploration in this field and inspire the development of advanced techniques. Researchers looking to integrate adversarial methods within their anomaly detection methodologies could find valuable insights and a starting point in this work.

6.3 Future Research

Potential future research avenues include the incorporation of the parameter value vector into AdvSVDD architecture. This addition holds promise for significantly enhancing the log anomaly detection capabilities of AdvSVDD.

Furthermore, a deeper exploration of the interaction between the anomaly detector and the adversary within AdvSVDD is warranted. Investigating optimal parameter configurations and training strategies could lead to further advancements in AdvSVDD's performance. Specifically, exploring variations in the dimensional structure of the two distinct LSTM components in AdvSVDD presents a compelling avenue.

In the context of weight value generation, future research might extend beyond the mean of the linear layer. Exploring alternative methods, such as using the max of the linear layer or employing more intricate functions, could result in more effective weight values. This exploration is geared towards enhancing the loss calculation between the

hidden representation of critical data points and the center of the hypersphere.

Domain-specific integration of AdvSVDD is another promising research direction. Investigating its performance in other domains with limited sequential data, such as in healthcare, could demonstrate the framework's efficacy compared to state-of-the-art models.

Additionally, expanding the application of AdvSVDD to dynamic, real-time datasets is a rich area for exploration. Future research could evaluate its performance in such contexts, providing insights into the potential and limitations of AdvSVDD in dynamic scenarios.

REFERENCES

[1] D. M. Hawkins, *Identification of outliers*. Springer, 1980, vol. 11.

[2] M. Ahmed, A. N. Mahmood, and M. R. Islam, "A survey of anomaly detection techniques in financial domain," *Future Generation Computer Systems*, vol. 55, pp. 278–288, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X15000023

[3] G. A. Susto, M. Terzi, and A. Beghi, "Anomaly detection approaches for semiconductor manufacturing," *Procedia Manufacturing*, vol. 11, pp. 2018–2024, 2017, 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978917305619

[4] M. Ahmed, A. Naser Mahmood, and J. Hu, "A survey of network anomaly detection techniques," *Journal of Network and Computer Applications*, vol. 60, pp. 19–31, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804515002891

[5] T. Fernando, H. Gammulle, S. Denman, S. Sridharan, and C. Fookes, "Deep learning for medical anomaly detection – a survey," 2021.

[6] S. Russo, M. Lürig, W. Hao, B. Matthews, and K. Villez, "Active learning for anomaly detection in environmental data," *Environmental Modelling & Software*, vol. 134, p. 104869, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364815220309269

[7] P. P. Pawar and A. C. Phadke, "A survey on different techniques for anomaly detection," in *International Conference on Computational Intelligence*. Springer, 2022, pp. 365–380.

[8] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," 2019.

[9] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[10] A. M. Kosek, "Contextual anomaly detection for cyber-physical security in smart grids based on an artificial neural network model," in *2016 Joint Workshop on Cyber- Physical Security and Resilience in Smart Grids (CPSR-SG)*, 2016, pp. 1–6.

[11] C. Zhang, X. Wang, H. Zhang, J. Zhang, H. Zhang, C. Liu, and P. Han, "Layerlog: Log sequence anomaly detection based on hierarchical semantics," *Applied Soft Computing*, vol. 132, p. 109860, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494622009097

[12] T. Zhang, A. Wiliem, S. Yang, and B. Lovell, "Tv-gan: Generative adversarial network based thermal to visible face recognition," in *2018 international conference on biometrics (ICB)*. IEEE, 2018, pp. 174–181.

[13] Y. Zhu, Y. Zhang, H. Yang, and F. Wang, "Gancoder: an automatic natural language-to-programming language translation approach based on gan," in *Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9–14, 2019, Proceedings, Part II 8*. Springer, 2019, pp. 529–539.

[14] I. Rasheed, F. Hu, and L. Zhang, "Deep reinforcement learning approach for autonomous vehicle systems for maintaining security and safety using lstm-gan," *Vehicular Communications*, vol. 26, p. 100266, 2020.

[15] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, "f-anogan: Fast unsupervised anomaly detection with generative adversarial networks," *Medical image analysis*, vol. 54, pp. 30–44, 2019.

[16] A. Geiger, D. Liu, S. Alnegheimish, A. Cuesta-Infante, and K. Veeramachaneni, "Tadgan: Time series anomaly detection using generative adversarial networks," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 33–43.

[17] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," 2018.

[18] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[19] P. Lahoti, A. Beutel, J. Chen, K. Lee, F. Prost, N. Thain, X. Wang, and E. H. Chi, "Fairness without demographics through adversarially reweighted learning," 2020.

[20] D. Jalal and T. Ezzedine, "Decision tree and support vector machine for anomaly detection in water distribution networks," in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, 2020, pp. 1320–1323.

[21] M. Chen, A. Zheng, J. Lloyd, M. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *International Conference on Autonomic Computing, 2004. Proceedings.*, 2004, pp. 36–43.

[22] B. Wang, S. Ying, G. Cheng, R. Wang, Z. Yang, and B. Dong, "Log-based anomaly detection with the improved k-nearest neighbor," *International Journal of Software Engineering and Knowledge Engineering*, vol. 30, no. 02, pp. 239–262, 2020.

[23] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016, pp. 207–218.

[24] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "Knn model-based approach in classification," in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and*

ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings. Springer, 2003, pp. 986–996.

[25] M. S. bin Sinal and E. Kamioka, "Early abnormal heartbeat multistage classification by using decision tree and k-nearest neighbor," in *Proceedings of the 2018 Artificial Intelligence and Cloud Computing Conference*, ser. AICCC '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 29–34. [Online]. Available: https://doi-org.dist.lib.usu.edu/10.1145/3299819.3299848

[26] B. Wang, S. Ying, and Z. Yang, "A log-based anomaly detection method with efficient neighbor searching and automatic k neighbor selection." *Scientific Programming*, pp. 1 – 17, 2020. [Online]. Available: https://dist.lib.usu.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=asn&AN=143540910&site=ehost-live

[27] G. de la Torre-Abaitua, L. F. Lago-Fernández, D. Arroyo, and S. Kotsiantis, "A compression-based method for detecting anomalies in textual data." *Entropy*, vol. 23, no. 5, p. 618, 2021. [Online]. Available: https://dist.lib.usu.edu/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=asn&AN=150474981&site=ehost-live

[28] F. Zhou, J. H. Park, and Y. Liu, "Differential feature based hierarchical pca fault detection method for dynamic fault," *Neurocomputing*, vol. 202, pp. 27–35, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231216300273

[29] K. Heller, K. Svore, A. Keromytis, and S. Stolfo, "One class support vector machines for detecting anomalous windows registry accesses," 12 2003.

[30] Y. Li, T. Zhang, Y. Y. Ma, and C. Zhou, "Anomaly detection of user behavior for database security audit based on ocsvm," in *2016 3rd International Conference on Information Science and Control Engineering (ICISCE)*, 2016, pp. 214–219.

[31] R. Vaarandi and M. Pihelgas, "Logcluster - a data clustering and pattern mining algorithm for event logs," in *2015 11th International Conference on Network and Service Management (CNSM)*, 2015, pp. 1–7.

[32] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1285–1298. [Online]. Available: https://doi.org/10.1145/3133956.3134015

[33] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," 2021.

[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[35] X. H. Le, H. Ho, G. Lee, and S. Jung, "Application of long short-term memory (lstm) neural network for flood forecasting," *Water*, vol. 11, p. 1387, 07 2019.

[36] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2018.

[37] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," 2019.

[38] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-supervised log parsing," pp. 122–138, 02 2021.

[39] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services (ICWS)*, 2017, pp. 33–40.

[40] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 4393–4402. [Online]. Available: https://proceedings.mlr.press/v80/ruff18a.html

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[42] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. null, p. 2121–2159, jul 2011.

[43] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.