Enhancing Insider Threat Detection Using User-Based Sequencing and Transformer Encoders

by Mohamed Elbasheer

Bachelor of Science in Econometrics and Social Statistics, May 1991,
University of Khartoum, Khartoum, Sudan
M.S. in Database technologies, August 2014, Regis University, Denver, Colorado

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial satisfaction of the requirements
for the degree of Doctor of Engineering

August 31, 2024

Praxis directed by

Adewale Akinfaderin
Professorial Lecturer in Engineering and Applied Science

The School of Engineering and Applied Science of The George Washington University certifies that Mohamed Elbasheer has passed the Final Examination for the degree of Doctor of Engineering as of July 26, 2024. This is the final and approved form of the Praxis.

Enhancing Insider Threat Detection Using User-Based Sequencing and Transformer Encoders

Mohamed Elbasheer

Praxis Research Committee:

Adewale Akinfaderin, Professorial Lecturer in Engineering and Applied Science, Praxis Director

Amir Etemadi, Associate Professor of Online Engineering Programs, Chair

Ould Vadel, Professorial Lecturer of Engineering and Applied Science, Committee Member

Dedication

To my mother, Insaf, whose love and guidance have made everything possible. To my wife Sawsan, my rock, and our children, Rammah, Sarah, Leena, and Ammar, who light up my life. To Dr. Hassan, whose brilliance inspires every endeavor. To Prof. Akinfaderin, whose wisdom and guidance shaped this journey. In memory of all my friends whose spirit inspires and guides me. To my family and friends, pillars of strength in every step of this journey. For all who have inspired me to seek knowledge and pursue my dreams. To Allah's Mercy, Guidance, and protection. May the essence of Adaptive Malicious Events Learners reflect even a fraction of your wisdom.

Acknowledgments

"He is the One Who sent down to you the Book, from which there are verses that are decisive in meaning — they are the foundation of the Book — and others that are allegorical. Those who have a twist in their hearts, they follow the allegorical verses, seeking controversy, and seeking its interpretation; but only Allah knows its interpretation. And those who are well grounded in knowledge say: 'We believe in it; it is all from our Lord.' And none take heed except the people of understanding." – Surah Aal-E-Imran (3:7).

I extend my heartfelt gratitude to my cherished wife, Sawsan, and our wonderful children, Rammah, Sarah, Leena, and Ammar. Your enduring patience and immeasurable sacrifices have been the cornerstone of this achievement. I am blessed to have such a supportive family by my side. A special gratitude to my parents, Zain Elabdeen and Insaf, and my siblings, Ahmed, Hamid, Mahdi, Elsamani, Amel, and Masharig, for their prayers and unwavering belief in me. Their support was the beacon of light guiding me through challenging times. My extended family's constant prayers, love, and encouragement have strengthened me tremendously. To the GWU community, I hope my journey serves as a testament to what is achievable and inspires those who follow in my footsteps. I'm grateful to my friends, whose continued encouragement and positive energy were invaluable throughout this journey. I want to express my appreciation to Oracle US and my esteemed colleagues. Many of you were instrumental in embarking on this journey, while others provided continuous support and encouragement. Your belief in me has been a driving force. To my Praxis Research Committee members, Dr.John Fossaceca and Dr. Shahryar Sarkani, thank you for your invaluable feedback and for endorsing my Praxis. I'm deeply indebted

to my advisor Dr. Adewale Akinfaderin, for his expert guidance and unwavering support. Above all, I humbly thank God. This achievement stands as a testament to His grace and guidance. Without Him, none of this would have been possible.

Abstract

Enhancing Insider Threat Detection Using User-Based Sequencing and Transformer Encoders

The increasing prevalence of insider threats and their costly impact have garnered considerable attention from researchers and security practitioners alike. According to the "The Cost of Insider Risk" report published by the Ponemon Institute in 2023, 71% of the surveyed companies reported experiencing between 21 and more than 40 insider incidents per year, representing a 4% increase from the same report in 2022. Unlike external intruders, insiders are more capable of executing attacks, often without advanced technical skills or experience. Detecting these attacks is particularly challenging due to the insider's authorized access to the system. Many machine-learning detection algorithms researchers have utilized to date have demonstrated varying results in identifying malicious insider activities. Although innovative, those algorithms typically view threat data as snapshots, often missing the long-term user behavior patterns.

In this study, we developed a novel User-Based Sequencing (UBS) method to restructure the CERT dataset into a sequential format and use a Transformer Encoder model to learn from normal user behavior. The output from the Transformer is then fed into three ML algorithms, One-Class Support Vector Machine (OCSVM), Local Outlier Factor (LOF), and Isolation Forest (IFOREST), to detect anomalies. Our Transformer model achieved state-of-the-art performance, with a high accuracy rate of 96.61%, 99.43% recall, an F1-score of 96.38%, a AUROC value of 95.00%, and a low False Negative Rate (FNR) and False Positive Rate (FPR) of 0.0057 and 0.0571, respectively.

The success of sequence-based models such as transformers is primarily credited to their innate capability to learn from the sequential representations embedded within languages naturally. Our proposed User-Based Sequencing aims to build those same sequential patterns on tabular data to ease the learning of temporal dependencies, thus allowing the models to adapt to new latent representations of users' behavior as their work patterns evolve. User-based sequencing should improve the detection accuracy of sequential models like transformers as well as non-sequential models like autoencoders. This assumption is validated in our experiment when we run the autoencoder on UBS versus tabular data. We saw an increase of 28.28%, 14.38%, and 30.77% in accuracy, precision, and recall, respectively. Thus, this research not only bridges the gap by employing the Transformer Encoders to effectively tackle the insider threat problem but also holds significant implications for future studies in the cybersecurity domain. Our novel User-Based Sequencing structure can serve as a robust foundation for developing advanced machine-learning models to address various cybersecurity threats.

Table of Contents

List of Figures

List of Tables

Glossary of Terms

ADASYN Adaptive Synthetic Sampling

AI Artificial Intelligence

ANN Artificial Neural Network

API Application Programming Interface

AUC Area Under the Curve

BERT Bidirectional Encoder Representations from Transformers

CERT Computer Emergency Response Teams

CISA Cybersecurity and Infrastructure Security Agency

CMU Carnegie Mellon University

CNN Convolutional Neural Network

CRTBP Circular Restricted Three Body Problem

CSV Comma-separated values

DBN Deep Belief Network

DL Deep Learning

DNN Deep Neural Networks

DTITD Digital Twin Insider Threat Detection

EDA Exploratory Data Analysis

ETL Extract, Transform, Load

FN False Negative

FNR False Negative Rate

FP False Positive

FPR False Positive Rate

GAN Generative Adversarial Network

GE Gray Encoding

GPT Generative Pretrained Transformer

GRU Gated Recurrent Unit

iFOREST Isolation Forest

IT Information Technology

ITD Insider Threat Detection

ITDM Insider Threat Detection and Mitigation

LOF Local Outlier Factor

LSTM Long Short Term Memory

MAITD Memory-Augmented Insider Threat Detection

MAML Model-Agnostic Meta-Learning

ML Machine Learning

MSELoss Mean Squared Error Loss

NETL Nuclear Engineering Teaching Laboratory

NITTF National Insider Threat Task Force

NLP Natural Language Processing

OCSVM OneClassSVM

RBM Restricted Boltzmann Machine

RF Random Forest

RNN Recurrent Neural Network

ROC Receiver Operating Characteristic Curve

SHAP SHapley Additive exPlanations

SMOTEENN Synthetic Minority Over-sampling Technique Edited Nearest Neighbors

SVM Support Vector Machine

TN True Negative

TP True Positive

UBB User-Based Batching

UBS User-Based Sequencing

Chapter 1: Introduction

1.1 Background

Insider threats pose a complex and challenging problem for organizations across government, public, and private sectors. The lack of sufficient data has made detecting such threats extremely difficult. However, over the last two decades, researchers have made significant progress in developing Machine Learning (ML) solutions that can effectively detect, prevent, and mitigate insider attacks. A 2023 report by the Ponemon Institute entitled "The Cost of Insider Risk" found that 64% of organizations worldwide with workforce ranging from 500 to over 75,000 employees consider ML to be an essential tool in addressing insider incidents (Ponemon, 2023). This highlights the industry's confidence in the power of Machine Learning to tackle this pressing problem.

1.2 Insider Threats

The Cybersecurity and Infrastructure Security Agency (CISA) defines an insider as "*any person who has or had authorized access to or knowledge of an organization's resources, including personnel, facilities, information, equipment, networks, and systems*" (CISA - Defining Insider Threats, 2023).

CISA characterizes an 'insider threat' as the potential for such actors to utilize their access or understanding of an organization to cause harm. This harm could manifest through various means, including malicious, negligent or unintentional actions that adversely affect the organization's integrity, confidentiality, availability, data, personnel, or facilities (CISA -

Insider Threat Mitigation Guide, 2020).

Furthermore, CISA outlines several manifestations of insider threats, ranging from espionage and terrorism to unauthorized information disclosure, participation in transnational organized crime, sabotage, workplace violence, and the deliberate or accidental compromise of organizational resources or capabilities.

Moreover, CISA categorizes insider threats into several types, including unintentional threats stemming from negligence or accidents and intentional threats, often called "malicious insiders." Among them, collusive threats involve insiders collaborating with external actors to harm the organization. In contrast, third-party insider threats arise from contractors or vendors who, though not formal organization members, have privileged access to essential assets and may pose direct or indirect threats (Devince, 2020).

Unlike external adversaries who encounter various barriers to entry, insiders inherently have legitimate access, placing them in a unique position to exploit the trust placed in them by the organization (Software Engineering Institute, 2022). Their intimate understanding of internal systems equips them to inflict significant, often undetected damage (Yuan & Wu, 2021).

An examination of the nature of insider threats reveals a range of motivations. At one end are the malicious insiders, individuals motivated by personal gain, financial incentives, or ideological beliefs (Maasberg et al., 2020a). Those actors misuse their privileged access for data theft, operational disruption, or espionage (Yuan & Wu, 2021). Their actions, concealed by their insider knowledge, often evade standard security measures, posing significant detection and prevention challenges (Glasser & Lindauer, 2013).

Equally concerning are the unwitting insiders, often characterized by unintentional misconduct (Software Engineering Institute, 2022). Through

negligence or lack of awareness, those individuals inadvertently aid malicious actors. For instance, they might click on phishing links or unknowingly share sensitive information, thus creating entry points for external attackers to compromise the organization's systems (Devince, 2020).

In 2023, the Ponemon Institute conducted an extensive study seeking to shed light on the significant implications of insider risks. This research gathered data-driven insights and responses from 1,075 Information Technology (IT) and IT security practitioners representing 309 organizations from across the globe (Ponemon, 2023). Table 1.1. summarizes the key findings from this study.

Table 1.1: *Key Findings from Ponemon 2023 Insider Risks Report*

Metric	**Value**
Average annual cost of insider risk (per-organization)	$16.2M
Increase in cost over four years	40%
Average Time to contain insider incident	86 days
Cost at 91 days of response time	$18.3M
Annual IT security budget allocated to insider risk management	8.2%
Percent of insider risk budget spent on post-incident activities	91.2%
Companies with insider risk programs started	77%
Insufficient funding for insider risk management	58%
Increase in insider risk management funding in 2024	46%
Importance of AI/ML in Insider Threat Detection (ITD)	64%

1.3 Research Motivation

The rise of insider threats within government, private, and public organizations has created an urgent need to find a solution. These threats can

come from authorized personnel, such as employees or contractors, and have become more significant as external cybersecurity measures have been strengthened (Greitzer et al., 2011). The potential financial, reputational, and legal implications of such threats are substantial "Gurucul" (2023a). To address this issue, the National Insider Threat Task Force (NITTF) has spearheaded efforts to develop a government-wide insider threat program for deterring, detecting, and mitigating insider threats (National Insider Threat Task Force, 2017). Many private and public organizations have followed similar initiatives to protect themselves. However, detecting insider threats poses significant challenges, particularly regarding threat-data availability and quality. Recent surveys suggest that advanced deep learning models such as LSTM, RNN, and Transformer can help overcome some of these data challenges by employing techniques such as Anomaly Detection (Yuan & Wu, 2021). Transformers, well-known for their ability to process large datasets and manage long-range dependencies using multi-head attention, offer a promising alternative to conventional machine learning models, which often struggle to process sparse and imbalanced data. Our study aims to leverage the unique capabilities of the Transformer to enhance Insider Threat Detection (ITD).

1.4 Problem Statement

A sharp rise in malicious insider threats has exposed U.S. organizations to increasing data breaches (Gurucul, 2023), Costing an average of 4.45 million dollars per breach (IBM Security, 2023).

In recent years, data security risks have increased for organizations in the United States. One significant contributor to this risk is the increased occurrence of malicious insider threats. SecurityMagazine's 2023 report

reveals a marked rise in insider threats, resulting in more frequent data breaches. These breaches have far-reaching implications beyond compromising organizational security, often leading to substantial financial losses. According to IBM's 2023 report (IBM Security, 2023), the average cost of a data breach is approximately 4.45 million dollars, encompassing both direct and indirect expenses. Direct costs include breach detection and management expenses, legal fees, and regulatory fines. Indirect costs include loss of customer trust, damage to an organization's reputation, and missed business opportunities.

1.5 Thesis Statement

A predictive model using Transformer Encoders and User-Based Sequencing will efficiently detect insider threat attacks and reduce corporate data breaches.

As the digital landscape continues to evolve, corporate threats are becoming increasingly sophisticated. Traditional methods for detecting data breaches may not always provide swift responses to counter insider threats. This study proposes a state-of-the-art approach to detecting insider threats, a particularly challenging task due to the perpetrators' privileged access. Our proposed model combines two key technological innovations: Transformer Encoders and User-Based Sequencing (UBS). Transformer Encoders, known for their exceptional performance in Natural Language Processing (NLP), excel in handling sequential data. Their adaptability and attention mechanism allow them to focus on relevant data segments, making them more versed in identifying patterns and anomalies than LSTM and RNN models. On the other hand, user-based sequencing transforms tabular data into a sequential format to enable the Transformer to achieve optimal

performance. We believe this integrated approach can effectively address insider threats and significantly decrease the incidence of corporate data breaches. Our main objective is to protect U.S. organizations from substantial financial, reputational, and operational losses by improving their ability to detect and address insider threats.

1.6 Research Objectives

This research aims to develop a predictive model to detect insider threats and protect U.S. organizations from insider data breaches. We plan to leverage advanced deep-learning models, such as Transformer and Autoencoder, built on publicly available datasets, such as the CERT r4.2, r5.2, and r6.2. We propose a unique user-based sequencing technique to bolster the accuracy of both Transformer and Autoencoder models in detecting this threat. Our ultimate goal is to significantly contribute to cybersecurity by creating robust and precise methods for detecting malicious insider threats.

1.7 Research Questions and Hypotheses

Detecting insider threats is a complex field that involves the intersection of behavioral, criminology, and psychology sciences. It presents significant challenges in information security. Our research is carefully designed to identify anomalies resulting from malicious insider activities, often characterized by subtle deviations from typical user behavior patterns. To guide our study, we formulate specific research questions and hypotheses to steer our investigation in the right direction and achieve our primary goals.

RQ1: How do Transformer encoders compare to Autoencoders in terms of accuracy, precision, and recall when detecting malicious insider threats?

RQ2: How does user-based sequencing improve the accuracy of insider threat detection?

RQ3: How can a model based on transformer encoders be optimized to enhance both speed and detection rate across various versions of the CERT insider threat datasets?

H1: Transformer encoders can capture long-range dependencies better than Autoencoders and accurately detect malicious insider threats.

H2: User-based sequencing enables the implementation of a Transformer Encoder on tabular data to improve the detection of insider threats.

H3: Using feature optimization will improve the speed and detection rate across various CERT dataset versions.

1.8 Scope of Research

Our research focuses on identifying insider threats in IT companies based in the United States using an unsupervised learning technique. Our objective is to develop and implement a novel Transformer Encoder model that leverages the Carnegie Mellon University (CMU)—Computer Emergency Response Teams (CERT) release r4.2 dataset. An important aspect of our approach involves integrating a novel user-based sequencing technique to convert tabular data into a format that aligns with the Transformer's attention mechanism. This should enable our model to effectively distinguish between normal and malicious users by learning from the long sequence of user activities. Our proposed model aims to demonstrate how Transformer encoders can be applied beyond Natural Language Processing to address the complex challenge of detecting insider threats.

1.9 Research Limitations

This study has a few limitations that affect the interpretation of its findings. One of the key limitations is that we relied on specific datasets from Carnegie Mellon University: CERT r4.2, r5.2, and r6.2. We chose these datasets because they are publicly available and designed to simulate real-world scenarios. However, they are synthetically created and might not entirely reflect real-world insider threat scenarios. Therefore, the extent to which our findings apply may be limited. Another challenge we faced was the significant imbalance in the dataset. Of the 1000 users in the CERT r4.2, only seventy were labeled as malicious. While this is sufficient for testing our detection rate, the small number of malicious users might have limited the depth of our analysis. Furthermore, the technical aspects of this research were constrained by the computing resources available to us. We used the PyTorch framework on personal computers and Google Collab Pro to develop and test our proposed Transformer. Using a different library or more powerful computational platforms might result in different outcomes.

1.10 Organization of Praxis

This study employs state-of-the-art technology, including Transformer encoders and User-Based Sequencing, to address the challenges of detecting insider threats. This study is organized into five chapters to achieve its goal. Chapter 2 examines the theoretical foundation of the research, exploring relevant literature on data preparation, feature optimization, and various machine learning models. Chapter 3 outlines the design and methodology of the proposed model, highlighting the anomaly detection techniques used and laying the foundation for subsequent chapters. Chapter 4 provides a

detailed account of the model implementation and performance, comparing and analyzing various baseline models, such as Autoencoders-Tabular and Autoencoder-UBS, to our primary Transformer model. Chapter 5 synthesizes and summarizes the research findings, reflecting on potential areas for improvement and outlining future research recommendations.

Chapter 2: Literature Review

2.1 Introduction

In recent years, the public media has reported several high-profile incidents of insider threats, notably the case of a former Tesla employee leaking thousands of personal records to a German news outlet in 2023, General Electric employees stealing trade secrets in 2020, Capital One data breach carried out by a single insider in 2019, and the disgruntled Tesla employee exfiltrating large amounts of highly sensitive data in 2018 (Gurucul, 2023b).

The increasing prevalence of insider threats and their costly impact have garnered considerable attention from both researchers and security practitioners (Safa et al., 2018). According to the "The Cost of Insider Risk" report published by the Ponemon Institute in 2023, 71% of the surveyed companies reported experiencing between 21 and more than 40 insider incidents per year. This is an increase of 4% from the same report in 2022 (Ponemon, 2023). Unlike external intruders, insiders are more capable of executing attacks, often without advanced technical skills or experience (Yuan & Wu, 2021). Detecting these attacks is particularly challenging due to the insider's authorized access to the system (Senator et al., 2013).

This chapter aims to examine the prevailing trend in insider threats research and pinpoint existing gaps. The primary objective of this review is to situate our proposed transformer-based model within the landscape of insider threat research and elucidate how our work integrates with and contributes to the existing corpus of knowledge.

2.2 Foundation of Transformer Encoders

This section introduces the transformer architecture, which is the primary algorithm used in this research.

The transformer offers a novel and effective alternative to the Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM), which were the predominant architecture used in many Insider Threat Detection (ITD) models. For instance, Randive et al. (2023) used Convolution Neural Network (CNN) to analyze an image representation of audit logs, while Alshehri (2022) transformed similar data into multivariate time series and applied RNN. Another study by AlSlaiman et al. (2023) employed LSTM to improve the insider detection rate using the CERT dataset. In addition, Hong et al. (2023) utilized an LSTM auto-encoder to uncover hidden patterns from sequential user activities, then combined a Graph Neural Network (GNN) and CNN with an organizational graph to detect malicious insiders. Moreover, Villarreal-Vasquez et al. (2023) proposed an LSTM-based anomaly detection system to address the insider threats data imbalance problem. Furthermore, Pal et al. (2023) suggested an ensemble approach using stacked-LSTM and Gated Recurrent Unit (GRU)-based attention models trained on single-day sequential activity logs to detect insider threats.

The transformer, developed by Vaswani et al. and published in the seminal paper "Attention Is All You Need" in 2017, is a groundbreaking development in the field of Natural Language Processing (NLP) and has gained widespread attention due to its exceptional performance in various language-related tasks (Vaswani et al., 2017). The transformer model features a classic encoder-decoder framework. In this setup, the encoder transforms an input sequence of symbols, denoted as $(x_1,\ldots,x_n)$, into a se-

quence of intermediate representations $\boldsymbol{z} = (z_1,\ldots,z_n)$. Subsequently, the decoder converts this sequence $\boldsymbol{z}$ into an output sequence $(y_1,\ldots,y_m)$. The model operates autoregressively at each step, meaning the output sequence depends on the prior sequence and a stochastic term. The encoder has six blocks (a tunable hyperparameter); each with two sublayers. The first is a multi-head self-attention structure, and the second is a position-wise fully connected feed-forward network including two linear layers activated by a ReLU function. The decoder is very similar to the encoder; however, it diverges from the encoder in two ways: the multi-head attention in the decoder uses keys & values from the encoder's output and queries from the preceding decoder layer. Additionally, the first self-attention layer in each decoder block is masked to prevent attention to subsequent positions, and the output embedding is right-shifted by one position to maintain autoregressive predictions (Wang & El Saddik, 2023). Figure 2.1 presents the transformer Encoders-Decoders architecture.

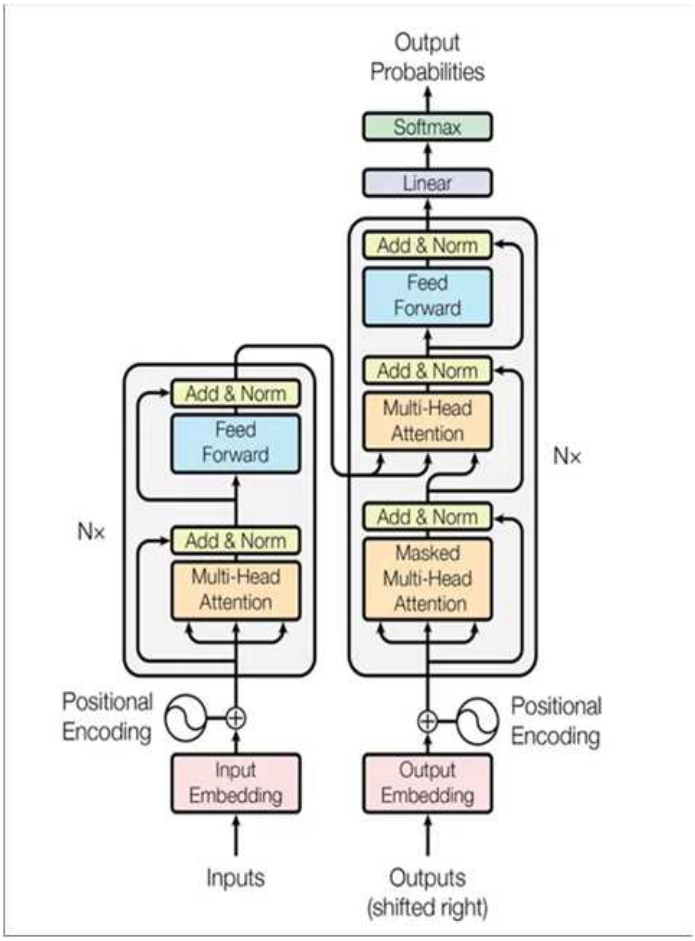


Figure 2.1: *Encoders-Decoders Architecture (Vaswani et al., 2017)*

In this research, we primarily focus on the encoder part of the transformer architecture. Our decision to choose the transformer encoder over the decoder is based on the common practice of using the encoder when the entire input sequence needs to be processed at once. Our user-based sequencing is designed to deliver the complete user data to the transformer, allowing the encoder to capture both short-term and long-term user activities. Nonetheless, the decoder architecture could also be adapted for this purpose with some modifications. For instance, BERT uses the encoder component of the transformer, while GPT-2 and GPT-3 use the decoder component, and both perform comparable tasks in NLP.

2.2.1 Encoder Architecture

The encoder has six identical blocks (a tunable hyperparameter), each containing two distinct sublayers. The first layer implements a multi-head self-attention mechanism, and the second is a position-wise, fully connected feed-forward network. Figure 2.2 presents the Encoder architecture.

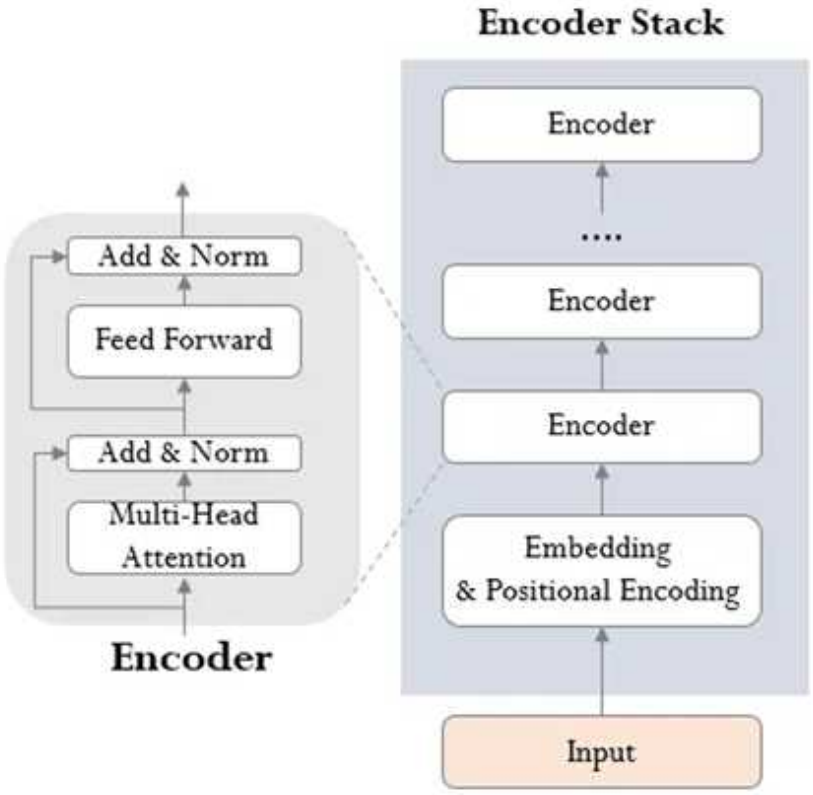


Figure 2.2: *Encoder Architecture - (Vaswani et al. 2017)*

1. **Input Embedding:** Embedding is the first step in transforming raw input data into a format that the transformer encoder can effectively process. The default transformer architecture uses the Word2Vec algorithm to convert the input into an embedding of a configurable vector size (512 by default).

2. **Positional Encoding:** A notable problem in the transformer archi-

tecture arises from its inability to inherently capture the order and position of elements within a sequence. This positional order is particularly important in natural language processing, where the position and arrangement of words play a significant role in conveying meaning. The transformer addresses this limitation by adding positional encoding into the input embedding to allow the model to learn the meaning of words based on their position in the sequence (Vaswani et al., 2017). Equation 2.1 and 2.1 show how positional encoding is implemented.

$$PE_{(pos,2i)} = sin(pos/1000^{2i/dm}) \tag{2.1}$$

$$PE_{(pos,2i+1)} = cos(pos/1000^{2i/dm}) \tag{2.2}$$

Where "PE" is the Positional Embedding, "pos" is position, "i" is the dimension, and d_m is the dimension of the input & output (Wang & El Saddik, 2023)

3. **Self-Attention Mechanism:**

The transformer's key innovation is its self-attention mechanism, which connects different positions within a sequence by allowing the model to weigh the importance of each position. This entire process begins with the input (embedding) being multiplied by three randomly initialized weight matrices: the query matrix (Q), the key matrix (K), and the value matrix (V). The attention is then calculated as a weighted sum of the values, where each weight is determined by a softmax function applied to the dot products of the queries and keys (Alammar, 2018). Furthermore, to stabilize training and prevent gradients from vanishing

or exploding, the dot product is divided by the square root of Query Vector.

Equation (2.3) from Vaswani et al.(2017) "Attention is all you need", shows how the self-attention is calculated.

$$SelfAttention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.3}$$

Unlike LSTMs and RNNs, which recurrently process the input sequences, the transformer employs a self-attention mechanism that simultaneously evaluates the entire input sequence (Gruetzemacher & Paradice, 2022). RNN and LSTM process the sequence one element at a time and rely on their hidden state to carry information from previously seen elements to the next steps (Sherstinsky, 2020). This is fine for small variable-length sequences but can lead to issues like vanishing and exploding gradients when the sequence is long (Sagvekar & Sharma, 2023). LSTMs are an advanced version of RNNs designed to mitigate some of these issues by introducing gates that regulate the flow of information. However, despite their enhanced capability in managing long sequences compared to RNNs, LSTMs are still constrained by their inherent sequential processing nature which prevents them from fully leveraging the modern parallel computing architectures. Furthermore, despite their improvements over RNNs, LSTMs may still encounter difficulties in handling extremely long sequences (Sherstinsky, 2020).

On the contrary, the transformer discards the sequential processing altogether. Instead of processing one element at a time, the transformer uses a self-attention mechanism to weigh the importance of different

parts of the input data relative to each other (Vaswani et al., 2017). This allows the model to consider the entire sequence simultaneously, making it very efficient for parallel computation and handling long-range dependencies. Additionally, the transformer employees multi-head attention to augment the self-attention. Rather than relying on a single set of attention weights, multi-head attention employs multiple sets, enabling the model to recognize various relationships within the data (Vaswani et al., 2017). This parallel processing capability helps the model to identify hidden patterns and dependencies within the input sequence better than LSTM and RNN (Yu et al., 2023). The transformer architecture has been highly influential as the foundation for many state-of-the-art models, such as Generative Pretrained Transformer (GPT) and Bidirectional Encoder Representations from Transformers (BERT).

2.3 Insider Threats Detection Approaches

Organizations tasked with protecting sensitive data or intellectual property are in constant battle to safeguard their assets from insider attacks. Traditionally, efforts to mitigate these threats relied on models based on psychological, behavioral, and decision-making theories (Agrafiotis et al., 2015; Greitzer & Hohimer, 2011; Legg et al., 2013; Nurse et al., 2014; Padayachee, 2016). However, despite their relevance, these methods can be subjective and complex to implement and scale due to privacy and ethical concerns and the ever-evolving nature of threats. Recently, there has been a significant pivot towards leveraging machine learning and deep learning techniques for more effective insider threat mitigation (Alsowail & Al-Shehari, 2020; Homoliak et al., 2019; L. Liu et al., 2018).

In the literature we review several recent surveys in which the authors extensively examined the different approaches used in detecting and mitigating insider threats. For instance, L. Liu et al. (2018) discuss malware and advanced persistent threats linked with insider activities. Homoliak et al. (2019) analyze various models and countermeasures employed in insider threat studies. Moreover, Alsowail and Al-Shehari (2020) provide a detailed analysis and comparison of various strategies employed by researchers, including classification algorithms, performance metrics, datasets, domains, and insider scenarios. Additionally,Al-Mhiqani et al. (2020) survey examine the widely used detection algorithms and machine learning techniques. They also categorize various types of insider threats, access methods, levels, motivation, reporting mechanisms, and proposing potential solutions.

Despite the growing number of studies on insider threats, the lack of real-world data has consistently hindered the exploration of effective countermeasures. The CMU-CERT dataset (Lindauer, 2020) is one of the most frequently used datasets for this purpose. However, it has a significant imbalance, with a disproportionately small number of positive (malicious) instances compared to a predominantly larger number of negative (non-malicious) cases. Building machine learning models on such disproportionally skewed datasets poses multiple challenges, particularly the potential for biased results toward the majority class. Many researchers have proposed various techniques to overcome this problem. For instance, Chattopadhyay et al. (2018) employed a cost-sensitive undersampling/oversampling technique, while Zou et al. (2020) used a variable weight-based approach to achieve the same goal. On the contrary, Pal et al. (2023) opted for an equally-weighted approach, while Randive and Ramasundaram (2023) chose to use the Synthetic Minority Over-sampling Technique combined with Edited

Nearest Neighbors (SMOTEENN). In contrast, Wang and El Saddik (2023) employed NLP for upsampling and Tomek Links for downsampling. Table 2.1 summarizes the commonly used techniques found in our literature review.

Table 2.1: *Class Imbalance Mitigation Techniques - CERT Dataset*

Reference	**Technique**
Chattopadhyay et al., 2018	Cost-sensitive adjustment
Zou et al., 2020	Weight-based
Nasser Al-Mhiqani et al., 2021	ADASYN
Al-Shehari & Alsowail, 2023	Random under/over-sampling
Pal et al., 2023	Equally-weighted
Asha et al., 2023	Nearmiss2 (NM-2)
Randive et al., 2023	SMOTEENN
Wang & El Saddik, 2023	Text augmentation & Tomek Links

2.3.1 Dataset

One of the major challenges facing cybersecurity researchers is developing and properly testing insider threat detection systems using real-world data. Unfortunately, finding relevant real-world datasets are scarce, mostly due to data privacy concerns and companies not willing to publish insider data for fear of public backlash (Bartoszewski et al., 2021; Yuan & Wu, 2021). The Computer Emergency Response Team (CERT) dataset, developed by Carnegie Mellon University (CMU), is one of the few publicly available synthetic datasets available for researchers (Lindauer, 2020). Despite its synthetic nature, the CERT dataset has been widely used in various studies including for example: Al-Mhiqani et al. (2021), Gayathri et al. (2020), Le and Zincir-Heywood (2021a, 2021b), and Le et al. (2020). A comprehensive

overview of the CERT dataset can be found in (Glasser & Lindauer, 2013).

2.3.2 Machine Learning Approaches

In today's digitally-driven society, organizations across various sectors, including technology firms, financial institutions, healthcare sectors, and government agencies, are undergoing an unprecedented digital revolution. This transformation, characterized by integrating digital technology in all aspects of business, fundamentally changes how organizations operate and deliver value to their customers (Schwertner, 2017). As a consequence, these organizations are generating data at an exponential rate. The sheer volume and variety of this data, coupled with its critical importance to business operations and its potential value to malicious insiders, significantly elevate the risks associated with insider threats. In this context, the detection and mitigation of insider threats using Machine Learning have become paramount for safeguarding organizational assets, intellectual property, and customer information. This perspective is supported by the works of (Al-Mhiqani et al., 2020; Homoliak et al., 2019; Qiang et al., 2018), who collectively argued that machine learning superior capabilities of quickly uncovering hidden anomalies in vast amounts of every-day activities positions it as an invaluable tool in this domain. Furthermore, a study by Smith (2018) highlighted the Machine Learning critical role in detecting early warning signs of insider threats. Additionally, a study by (Kabeireho, 2020; Olano et al., 2014) emphasized the vital role machine learning plays in educational programs, providing insights into risky behaviors and facilitating risk mitigation. This perspective is corroborated by the Ponemon Institute's 2023 report, which indicates that non-malicious users were responsible for 75% of insider threats (Ponemon, 2023).

2.3.2.1 Machine Learning Techniques and Methods

Historically, insider threat detection has relied on profiling and monitoring for anomalous behaviors using rule-based methods to scrutinize user actions (Greitzer et al., 2019). For instance, Legg et al. (2017) developed a tree-structure method for analyzing user activities across job roles to detect deviations from normative behavior patterns, while Nguyen et al. (2003) and Hanley and Montelibano (2011) employed signature-matching and rule-based systems for detecting unauthorized access and data usage anomalies. However, as noted by AlSlaiman et al. (2023), rule-based methods face scalability challenges, requiring frequent rule updates to keep pace with evolving threats, which increases maintenance efforts and the risk of false positives.

To address the shortcomings of rule-based approaches, classification-based models such as Decision Trees (Janjua et al., 2021), Random Forest (RF) (Nasser et al., 2022), Logistic Regression (Maasberg et al., 2020b), Naive Bayes (Janjua et al., 2021), Support Vector Machines (SVMs) (Asha et al., 2023), K-Nearest Neighbors (Kim et al., 2020), AdaBoost (Mehmood et al., 2023), XGBoost (Zou et al., 2020), LightGBM (Mehmood et al., 2023), Artificial Neural Networks (ANN) (Williams et al., 2022), and Deep Neural Networks (DNN) (Al-Mhiqani et al., 2021) have become increasingly popular in recent years. However, classification models grapple with the scarcity and severe imbalance of the existing insider datasets. To counteract this problem, researchers often use techniques like oversampling, undersampling, and data augmentation to manage dataset imbalances (Al-Shehari & Alsowail, 2023). Although these strategies can address some of the imbalance issues, they introduce complexities in data preprocessing. For example, selecting the appropriate sampling technique and identifying the optimal sampling

rate tailored to the dataset's specific traits and chosen algorithm presents a considerable challenge (chung Liu, 2004). Improper application of sampling techniques may result in models that perform well in test settings but falter in real-world applications (Al-Shehari et al., 2023; Randive & Ramasundaram, 2023).

In recent years, anomaly-based detection methods have become a popular alternative for insider threat detection. This is supported by various studies such as (Al-Shehari et al., 2023; Singh et al., 2023; Bartoszewski, 2022; Haidar & Gaber, 2018; Jang et al., 2020b; Le & Zincir-Heywood, 2021a; Randive & Ramasundaram, 2023; Villarreal-Vasquez et al., 2023; Xiao et al., 2023; Yao et al., 2018). Anomaly-based approaches offer an advantage over signature and classification-based methods as they are usually not restricted by predefined rules or dependent on a balanced data distribution (Lo et al., 2018). However, as noted by (Tae-Young & Cho, 2018), anomaly-based models are prone to a higher rate of false positives due to the incredibly broad definition of "normal." This broadness can result in behaviors that are "normal" but unusual being misclassified as anomalies (Yao et al., 2018). This issue is more pronounced in dynamic and complex environments, particularly where the training data fails to cover the broad spectrum of normal behavior (Amer & Abdennadher, 2011).

2.3.3 Deep Learning

As cyber threats grow increasingly complex and sophisticated, researchers in cybersecurity have turned to deep learning as it becomes challenging for traditional "shallow" machine learning to detect these threats effectively (Milosevic & Ciric, 2022). Deep learning is an advanced branch of machine learning that uses multi-layer neural networks to learn and extract insights

from a wide range of data, including time series, text, sequences, images, video, and audio files (Chollet, 2021). According to Ahmad et al. (2021), around 60% of intrusion detection methods proposed in recent years have been based on deep learning approaches, while 20% have been hybrid methods combining machine learning and deep learning. Despite their popularity, deep learning models often require large amounts of labeled data for training and significant processing power and memory, making them less accessible to individuals and organizations with limited computational resources (Yuan & Wu, 2021). Moreover, adding multiple layers, while beneficial for complex data pattern recognition, exacerbates the challenge of model interpretability. Nonetheless, the declining cost and increased availability of advanced computational hardware have recently facilitated the broader adoption of deep learning models in both supervised and unsupervised learning tasks, as detailed by (Goodfellow et al., 2016).

Although traditional machine learning techniques like classification and anomaly detection have been successful in various cybersecurity applications, they have their limitations. For instance, classification models rely heavily on labeled data, which can limit their effectiveness in scenarios with poor data quality or insufficient data (Heejung & Hwankuk, 2022). In contrast, anomaly-based methods tend to generate a high rate of false positives (Singh et al., 2023), which can overwhelm security analysts with unnecessary alerts and obscure genuine threats (Gill, 2012). On the other hand, deep learning models offer a promising solution to these challenges. They excel at deep feature synthesis (Kanter & Veeramachaneni, 2015), which automates feature engineering by enabling models to extract complex, hidden representations and features directly from large datasets. Unlike shallow machine learning techniques, deep learning eliminates the need for

manual preprocessing and feature selection (Goodfellow et al., 2016). This capability represents a paradigm shift in analyzing and interpreting data for threat detection (Ahmad et al., 2021; Yuan & Wu, 2021). Deep learning also simplifies the model-building process by removing human intuition and prioritizing the most salient features through the network architecture (Chollet, 2021; Kanter & Veeramachaneni, 2015). However, it is crucial to note that while deep learning models are valuable tools, they are not always the perfect solution for every problem. They can be computationally expensive and require substantial data to train (Bartoszewski, 2022).

In the course of our review, we came across a diverse array of machine learning models and techniques proposed for tackling the problem of insider threats. Predominantly, these models are based on deep learning networks. This trend highlights a growing consensus among researchers regarding the pivotal role of deep learning in mitigating the challenges posed by insider threats. For instance, Randive et al. (2023) introduced a Wavelet Convolutional Neural Network (WCNN) that converts user activity logs into image-based features through spectral and spatial analysis for insider threat classification. This approach achieved impressive results on the CERT dataset, with high accuracy (97.19%), AUC (97.30%), and minimal false positives. Furthermore, Dongyang et al. (2021) proposed an image-based that converts audit logs into grayscale images and uses geometric transformations to create a self-labeled dataset. IGT showed superior performance compared to traditional autoencoder-based approaches, with a 4% improvement in instance-based and a 2% improvement in user-based AUROC scores.

In contrast to image-based approaches, Alshehri (2022) developed a novel RNN model that processes audit data as a multivariate time series

to identify inter-relationships between activities. This model showcased an impressive AUC of 99% on the CERT dataset, but it underperformed in F1, precision, and recall, with scores of 80%, 99.12%, and 67.12%, respectively.

Another study by AlSlaiman et al. (2023) utilized sentiment analysis combined with Gray Encoding to differentiate between benign and malicious user behavior. Their methodology achieved a low False Positive Rate (FPR) of 0.29%, a False Negative Rate (FNR) of 2.47%, and an Area Under the Curve (AUC) of 97%. Despite these promising results, using Gray Encoding while simplifying data representation could lead to a loss of detail, potentially impairing the effectiveness of model training (Wang et al., 2022).

Furthermore, a study by Soh et al. (2019) developed a novel framework incorporating Gated Recurrent Units (GRUs) and skipgram techniques for early detection of insider threats. This approach employed aspect-based sentiment analysis and social network data for profiling and ranking employees. However, the evaluation was conducted on an augmented Enron email dataset, which is considered outdated and less frequently utilized in current research.

On the other hand, Zhang et al. (2018) aimed to improve insider threat detection by implementing a Deep Belief Network (DBN) with adaptive optimization of its network structure. They applied dichotomy and the golden section method to fine-tune the DBN parameters, including batch size, Restricted Boltzmann Machine (RBM) iterations, and the number and depth of hidden layers. This tailored approach resulted in a detection rate of 97.872% on the CERT dataset.

2.3.4 Autoencoders

Autoencoders, a unique type of neural network known for effective data representation, are increasingly utilized in anomaly detection beyond their traditional roles in dimensionality reduction and feature learning (Taormina & Galelli, 2018). For instance, Chen et al. (2012) investigated autoencoders in anomaly detection for collaborative information systems, while Yin et al. (2017) integrated them with RNNs to improve intrusion detection. Additionally, Taormina and Galelli (2018) demonstrated their efficacy in detecting cyber-attacks on water distribution systems, highlighting their adaptability and effectiveness in diverse applications. Many recent studies highlight the effectiveness of autoencoders in detecting insider threats. Jang et al. (2020a) employed a sequence-to-sequence autoencoder and RNN, achieving an AUC of 98.55% . Conversely, Chattopadhyay et al. (2018) utilized a scenario-based approach with time-series data to analyze user behavior and detect insider. Their model outperformed traditional classifiers like Random Forest and Multilayer Perceptron with an F1 score of 96.06% F1.

Moreover, Wei et al. (2021) crafted an end-to-end framework for predicting insider threats, utilizing cascaded autoencoders (CAEs) and a joint optimization network. This framework outperformed current unsupervised methods, achieving a low False Positive Rate (FPR) of 0.051, a Recall of 92.%, and an AUC score of 93.2%. Conversely, Dongyang et al. (2022) presented a novel Memory-Augmented Insider Threat Detection (MAITD) model designed to improve detection in organizational systems by integrating individual and group models to capture both temporal and spatial correlations in user behaviors, effectively reducing the false negative rate. Nonetheless, its detection and precision rates remain moderate compared to other autoencoder models.

2.3.5 Transformer Encoders

The transformer, unlike LSTMs, does not use recurrence or requires any sequential computation. This means that the transformer is not affected by the vanishing gradient problem (Sagvekar & Sharma, 2023). While attention mechanisms existed before the transformer, it was the transformer that made practical use of attention in a novel and powerful way (Gruetzemacher & Paradice, 2022). The transformer architectural advantage has led to significant breakthroughs in fields beyond NLP, such as computer vision, (Khan et al., 2022), autonomous driving (Prakash et al., 2021) , Intrusion detection system (Wu et al., 2022), and even Twitter sentiment analysis (Naseem et al., 2020). In addition, studies such as the ones in (Preeti et al., 2022) and (Hoang et al., 2021) have showcased the potential of transformer in anomalies detection using multivariate and time-series sequences. However, despite the broad adoption of transformer frameworks across various domains, a discernible void in their utilization for insider threat detection has been identified through our literature review.

During our literature review, we came across an impressive study by Wang and El Saddik (2023), which is highly relevant to our own research because it leverages the transformer and uses the CERT dataset — a rare combination— which should allow us to fairly compare and contrast their work with our proposed model.

In their study, Wang and El Saddik (2023) introduce the Digital Twin Insider Threat Detection (DTITD) framework, which uses Digital Twin technology and the transformer along with other deep learning techniques to enable real-time insider threat detection. The framework consists of three main components: data preparation, model construction, and the Insider Threat Detection module. Initially, their framework uses an Extract, Trans-

form, Load (ETL) pipeline to collect real-time log data, including login, device, file, email, and HTTP logs. The raw data then undergoes four preprocessing stages: cleaning, merging, feature selection, and grouping/ordering. Once all four stages are complete, the final results are stored in a database, and their digital twin properties are updated in the cloud. The framework uses scenario-based definitions and customized user-defined parameters to trigger the model training pipeline. This process is further tuned by learning additional parameters from the users' historical behaviors using deep learning. Once the model is trained, the inference pipeline is periodically activated to process the real-time log data. Subsequently, the detection results are recorded in a database table, and any identified anomalies trigger the creation of an incident alert. Because the CERT dataset is notorious for its imbalance, their study relies heavily on downsampling, upsampling, and text augmentation strategies, which our proposed model entirely bypasses.

Wang and El Saddik (2023) evaluated their custom-built transformer model, DistilledTrans, along with some pre-trained models such as BERT and RoBERTa, as well as a hybrid CNN-LSTM architecture on the CERT r4.2 and r6.2 datasets. While their results were robust, they were easily outperformed by our proposed model. Their original transformer model achieved an AUC of 97.37%, an accuracy of 93.53%, a precision of 98.13%, a recall of 80.71%, and an F1-score of 88.61%, which were lower than those achieved by other baseline models like RoBERTa-LSTM and LSTM-AutoEncoder models. However, their custom DistilledTrans model out performed all baselines, achieving a slightly higher recall of 84.62% and an F1 score of 90.53%. Nonetheless, this is still less than our model results of 99.43% recall and 96.38% F1-score.

In a different study, the TRANSLOG framework, developed by a team led

by Guo et al. (2021), leverages the transformer architecture to detect anomalies in log files. Their framework consists of two stages: pretraining and adapter-based tuning. In the pretraining stage, TRANSLOG acquires shared semantic knowledge of log data and then transfers the pre-trained model to the target domain through adapter-based tuning. Adapter-based tuning is a widely used technique to reduce the computational resources needed for training the complete model (Bapna et al., 2019). TRANSLOG demonstrates superior performance over traditional baseline models such as SVM, LR, Deeplog, and others across three benchmark datasets: Thunderbird, HDFS, and BGL. Notably, it achieved outstanding results on the HDFS dataset, with a precision of 0.999%, recall of 0.996%, and an F1 score of 0.998%.

Additionally, a recent study by Tuli et al. (2022) introduced TranAD, a state-of-the-art deep transformer network model designed to detect anomalies in multivariate time-series data. TranAD uses attention-based sequence encoders for inference and focused score-based self-conditioning to enhance feature extraction. The authors employed adversarial training to ensure model stability and integrated Model-Agnostic Meta-Learning (MAML), to increase training efficiency in limited datasets. Empirical evaluations of TranAD underscore its superiority over established methods like Merlin, KLSTM-NDT, DAGMM, and OmniAnomaly, achieving a 17% improvement in F1 scores and a significant 99% reduction in training time. However, it is noteworthy that TranAD has not been tested on the CERT dataset, primarily utilizing time-series data from various sources.

Furthermore, Li et al. (2022) proposed MalTransEn, a transformer encoder-based classifier to process long API call sequences in order to increase the detection precision of malware using a benchmark dataset created by matching the Windows API call sequences and eight malware classes. Their model

addresses the vanishing gradient limitations using RNN/LSTM models when dealing with long sequences. Their empirical findings demonstrated that MalTransEn significantly outperformed various deep learning baseline models such as LSTM and GRU, showcasing superior accuracy, precision, recall in malware detection.

The findings from the above pioneering studies such as DTITD (Wang & El Saddik, 2023), TRANSLOG (Guo et al., 2021), TRANAD (Tuli et al., 2022), and Li et al. (2022) highlight the effectiveness of transformer-based methods in anomaly detection using different dataset and domains. These findings corroborate the validity of our proposed transformer-based anomaly detection approach.

2.4 Summary

This chapter outlines the application of machine and deep learning in ITD, emphasizing the need for both reactive and proactive strategies. It discusses the challenges of data imbalance and the effectiveness of machine learning methods in addressing them. Our review identifies deep learning as a promising solution in this regard, primarily through their advanced deep feature synthesis and pattern detection. However, our review also identifies a significant gap in the literature: the potential of transformer architectures, which have shown remarkable success in various fields, remains largely unexplored in the context of ITD. Our research seeks to fill this gap by leveraging the transformer architecture and a novel user-based sequencing approach for enhanced threat detection. This integration aims to improve detection accuracy and efficiency, offering a significant contribution to cybersecurity.

Chapter 3: Methodology

3.1 Introduction

This chapter presents the methodology to address the research questions and hypotheses outlined in Chapter 1. Conceptually, we employ an insider threat anomaly-based detection approach using a novelty strategy. In this context, "Novelty" means the training dataset is devoid of any anomalies, enabling the model to learn and establish a baseline of "normal" user behavior. Our model predictive pipeline is depicted in Figure 3.1.

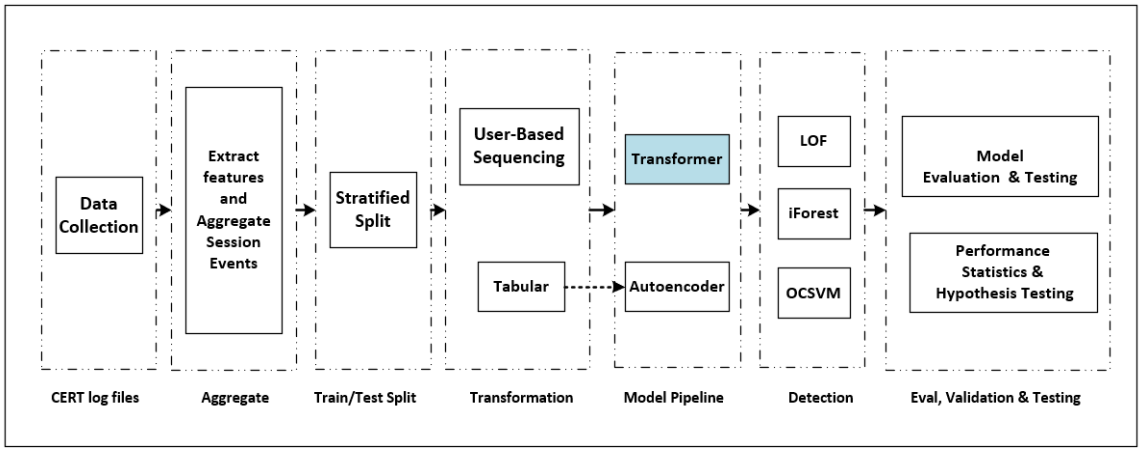


Figure 3.1: *Model Predictive Pipeline*

Insider threats can cause significant damage to organizations, making insider threat detection an important aspect of security for many industries (Ponemon, 2023). As detailed in Chapter 2, researchers often employ various techniques to tackle insider threat problem, including classification-based (Janjua et al., 2021), anomaly-based (Yao et al., 2018), signature-based (Nguyen et al., 2003), and rule-based methods (Greitzer et al., 2019). However, anomaly-based methods are widely recognized as a prevalent approach,

particularly when labeled data is scarce or non-existent (Singh et al., 2023; Haidar & Gaber, 2018; Villarreal-Vasquez et al., 2023). Figure 3.2, adapted from Gyanchandani et al. (2012), presents a taxonomy of existing anomaly-based insider threat detection systems.

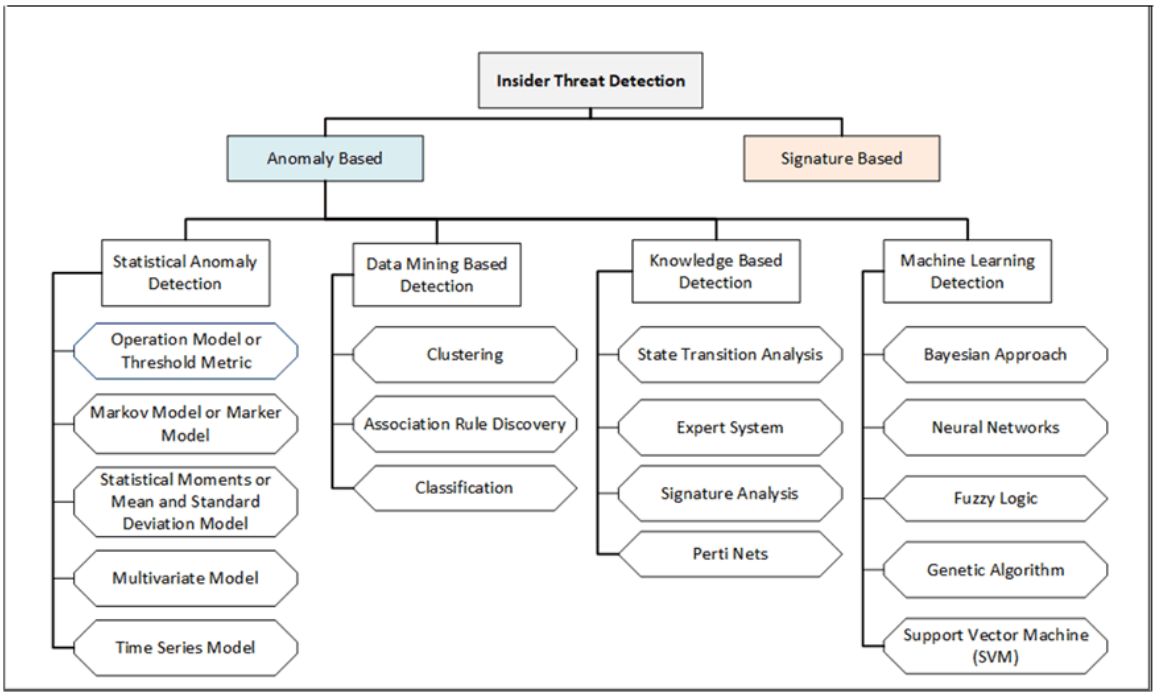


Figure 3.2: *Taxonomy of Anomaly Detection Methods*

Anomaly detection, closely associated with outlier detection, involves identifying unusual events or observations that stand out from the norm. The distinction can be conceptualized as follows: anomalies represent genuinely unusual or unexpected events, whereas outliers are simply data points that fall outside the normal range of values. For instance, a value significantly higher or lower than others might be labeled an outlier. In contrast, a data point indicating a measurement error or a fraudulent transaction would be deemed an anomaly (Bartoszewski, 2022). The boundary between anomalies and outliers is often blurred, leading to their interchangeable use. Both terms broadly encompass identifying unusual events or observations.

The chosen identification method often hinges on the data characteristics and desired outcome (Orabi, 2023).

In their survey on outlier detection, Gogoi et al. (2011) identified three types of anomalies. Point anomalies are individual data points that deviate significantly from the rest. Contextual anomalies take into account both the context and behavior to identify what is unusual. On the other hand, Collective anomalies involve groups of data points that, when viewed together, signify something unusual. Typically, existing anomaly detection methods focus on one or two of these types. They often neglect the long-term context of the user behavior by partially focusing on a subset of user action. For instance, the anomaly approach proposed by Bartoszewski (2022) is limited to examining only 21 days worth of user data. Conversely, the LSTM detection framework proposed by Villarreal-Vasquez et al. (2023) emphasizes filtering events on a per-application basis. With this in mind, we chose to use a long-term anomaly detection approach by analysing user (employee) data from first day of employment to their last. We hypothesize that this would help to thoroughly examine user behavioral patterns and trends than would a brief snapshot of their activities.

3.2 The Environment

The specification of the system used for code development and data processing in this research is a dual-boot system running Windows 10 and Ubuntu 20.04 LTS. The hardware is a Dell Precision 1750 workstation with 128GB RAM, 6TB SSD storage, and an 8GB GPU. The software is Python 3.9 and PyTorch framework version 2.0.1+cu118 (Paszke et al., 2019).

3.3 About the Dataset

The CERT dataset, created by Carnegie Mellon University, is one of the most widely used datasets in insider threat research. It is a synthetic dataset and publicly accessible (Lindauer, 2020). The dataset mimics the log behaviors of a virtual organization and includes user (employee) activity logs of 1000 insiders tracked over 17 months from January 2010 to May 2011. The dataset is available in several releases, created at different times, and includes multiple versions (e.g., r4.2, r5.2, r6.2, etc). Version r4.2 is classified as a "dense" dataset featuring numerous insider and malicious activities. Conversely, r6.2 is classified as a "sparse" dataset, containing records of only five insiders and 3995 regular users. The most frequently used version is r4.2 and the less used is r6.2 (Wang & El Saddik, 2023). Table 3.1 provides a statistical summary of CERT r4.2, r5.2, and r6.2 datasets.

Table 3.1: *CERT release r4.2 and r6.2 statistics*

Version	Employees	Insiders	All Activities	Malicious Activities
4.2	1,000	70	32,770,227	7,323
5.2	2,000	99	79,856,664	10,306
6.2	4,000	5	135,117,169	470

In this study, we used version r4.2 to train our model. The rationale behind selecting version r4.2 is that it has the second largest number of malicious users and activities, 70 and 7323, respectively, and it is one of the widely used versions in research (Wang & El Saddik, 2023). However, we are not restricted to using r4.2. Our model is designed to be used with any dataset that has a similar structure to the CERT dataset. The

only requirement is to transform the dataset using our novel User-Based Sequencing (UBS) structure.

3.4 Model Pipeline

Considering the dataset consists of data in raw log format, the first decision we faced was choosing between three levels of granularity, session, daily, or weekly. To extract the correct features and build the appropriate data representation for our model, We elect to use a session-based granularity and construct fixed-size vectors consisting of 35 encoded categorical and numerical features. Each numerical feature tracks the count of how many times a specific event was performed. We believe that a session-based granularity approach enables capturing subtle changes in user behavior that may go unnoticed when the data is aggregated over an extended period.

Secondly, we needed an approach to transform the extracted tabular data into a sequential format since this is the requirement for using the Transformer by design (Child et al., 2019). Therefore, we built a novel User-Based Sequencing technique to convert the extracted feature vector into a nested Python dictionary structure. A Python dictionary is a collection that associates keys with corresponding values and is recognizable by its curly brace "{}" notation (Novak, 2019). In our case, the keys represent employee IDs, while the values consist of three tuples. The first tuple denotes the day of the week corresponding to the dataset tracked days (1-501). In contrast, the second tuple is designed to accommodate the maximum number of sessions per day (9 in our experiment), and the third tuple tracks the 35 captured features and their values. With this multi-index structure, we aim to tokenize all user activities sequentially and enable the Transformer to efficiently track the long-term dependency between days, sessions, and

features.

Thirdly, we use our novel UBS structure to train the Transformer model on only benign data and evaluate its performance based on its ability to reconstruct its input, with any deviations from the expected output flagged as potential anomalies (Lee & Kang, 2022). Finally, instead of using conventional statistical methods like mean, standard deviation, and Median Absolute Deviation (MAD), which can be skewed by data distribution and outliers. We elect to use unsupervised machine learning algorithms — Local Outlier Factor (LOF), One-Class Support Vector Machine (OCSVM), and Isolation Forest (iForest)—to evaluate the reconstruction errors. This approach provides a more precise and accurate method for identifying anomalies by leveraging the strengths of these algorithms to overcome the limitations of statistical techniques. Our proposed model pipeline is summarized in Figure 3.3 and the detailed system overview is depicted in Figure 3.4.

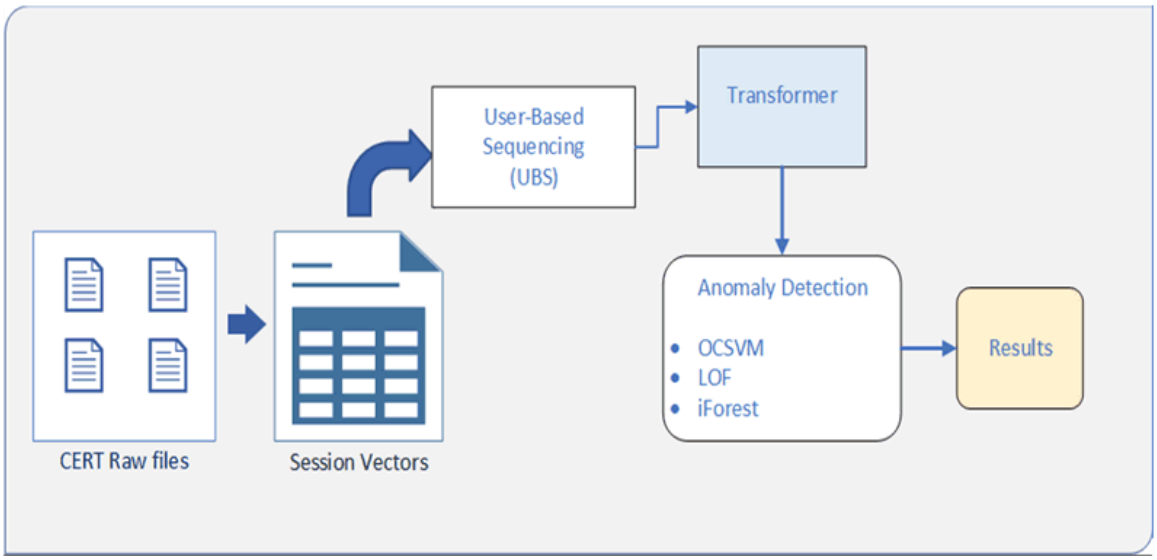


Figure 3.3: *Methodology Pipeline*

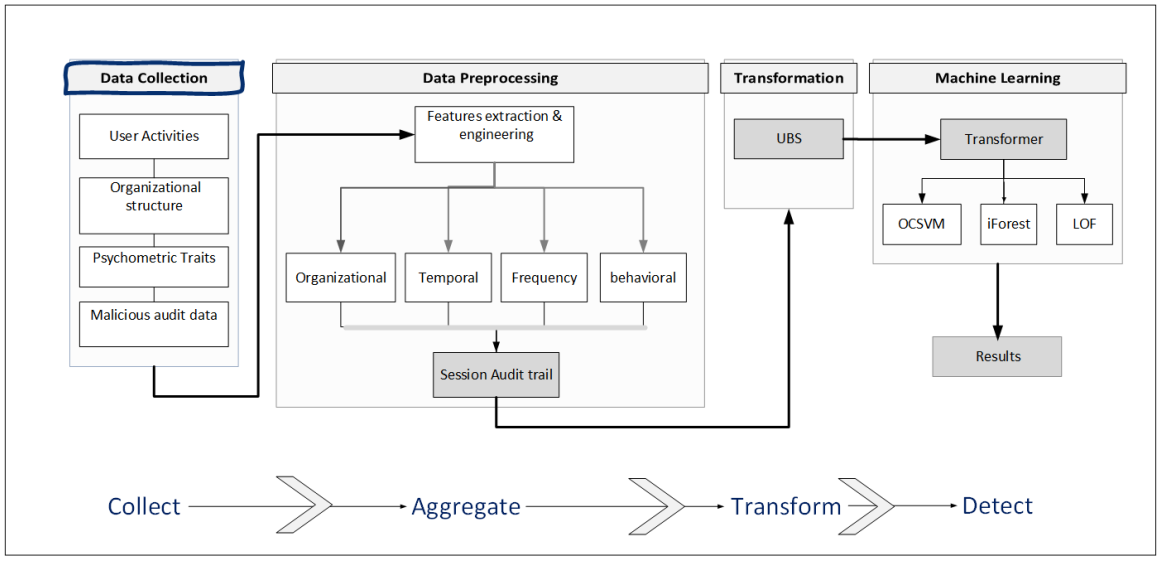


Figure 3.4: *System Overview*

3.5 System Architecture

The proposed system, illustrated in Figure 3.4, operates through the following stages:

3.5.1 Data Collection

We processed several CERT raw data sources for our model data pipeline, including web history, email, device access records, logon activities, file access events, and user psychometric and organizational data. The following section details the features captured from these sources before undergoing data correlation and aggregation.

logon.csv: Tracks user logon activities, including event ID, timestamp, user ID, PC ID, and activity type (logon or logoff).

email.csv: Captures email interactions, including details like event ID, timestamp, user ID, PC ID, recipients (to, cc, bcc), sender (from), activity type (Send, Receive, or View), size, attachments, and content.

http.csv: Records web activity, including event ID, timestamp, user ID, PC ID, webpage URL, and activity type (such as "WWW Visit", "WWW Download", "WWW Upload"), along with the content involved.

file.csv: Tracks file operations, including event ID, timestamp, user ID, PC ID, filename, and activity type (open, copy, write, delete). It also specifies interactions with removable media and content

device.csv: logs device connection activities, including event ID, timestamp, user ID, PC ID, file-tree, and activity type (connect or disconnect).

psychometric.csv: The psychometric log presents the results of personality evaluations for all employees, utilizing the Big Five OCEAN model.

insiders.csv The file contains an overview of attack scenarios in all CERT dataset releases.

LDAP: The LDAP (Lightweight Directory Access Protocol) consists of 18 files, each corresponding to a monthly record. These files furnish comprehensive organizational information about the enterprise, including details such as employee names, supervisor connection, and departmental designations.

The CERT version r4.2 contains over 32 million events from 1,000 users' activity logs over 501 days, including 70 users who executed attacks in three different scenarios. Table 3.2 summarizes the dataset's statistics, and Table 3.3 details the three attack scenarios.

Table 3.2: *CERT r4.2 event counts*

Event Type	**No. of Events**
Log in & out	854,860
Using removal media	405,380
Email Traffic	2,629,980
WWW Traffic	28,424,429
File Operations	445,581

Table 3.3: *CERT r4.2 Attack Scenarios*

Scenario	**Description**
1	User who did not previously use removable drives or work after hours begins logging in after hours, using a removable drive, and uploading data to Wikileaks.org. Leaves the organization shortly after.
2	User begins surfing job websites and soliciting employment from competitor. Before leaving the company, they use a thumb drive (at markedly higher rates than their previous activities) to steal data.
3	System administrator becomes disgruntled. Downloads a key-logger and uses a thumb drive to transfer it to his supervisor's machine. The next day, he uses the collected keylogs to log in as his supervisor and send out an alarming mass email, causing panic in the organization. He leaves the organization immediately.

All the CERT dataset versions include various insider threat scenarios, such as unauthorized access, data exfiltration, theft of intellectual property, and sabotage. However, in this study, we have decided to use a binary classification to identify the user activity as either malicious or benign. Therefore, we have combined all the malicious scenarios mentioned above and encoded them as '1', while benign activities are labeled as '0'.

3.5.2 Data Pre-processing and Aggregation

Because all the CERT data sources outlined in the previous section are presented in Comma-separated values (CSV) files, they require parsing and preprocessing to extract any useful data associations. We start by looping through the raw data and extracting information such as Logon and Logoff timestamps, device association, file access, email interaction, and HTTP activities. Furthermore, additional aggregation is performed to engineer additional statistical features across various activities based on timestamps, number of sessions, HTTP domains visited, and internal and external email communications. To help in this endeavor, we modified and used the scripts from (Le et al., 2020) to simplify this process and not reinvent the wheel. These scripts process and create auxiliary information about each user, such as assigned machines, relationships to other colleagues, roles, working hours, number of concurrent sessions per user, session duration, HTTP categorization, and device and file usage.

Since the CERT dataset does not make any assumptions about working hours, PC ownership, and HTTP categorization, we must decide how to set this information. Guided by previous research, we set working hours for weekdays, weekends, and holidays according to applicable rules in the United States. For instance, we consider the time between 7:30 and 17:30 to be working hours on weekdays (Zhenjiang et al., 2021). Furthermore, websites are categorized according to their contents, such as job-hunting, hacktivist activities, and file-sharing, and the user-assigned PC is identified based on the device they typically use.

3.5.3 Feature Extraction

The feature extraction process consists of two principal stages. The first stage commences after compiling the data from the CERT source files. We used the user ID and a specific aggregation condition c, such as the time and type of actions performed, to extract features and produce numerical vectors X_c, referred to as data instances. These vectors are of a fixed length, N, and encapsulate all user actions. In the second stage, we populate the data instance using the aggregation condition from the first stage and label the activity as either malicious or benign according to the information furnished in the CERT "insiders.csv" file (Singh et al., 2023). Categorical features—such as PC identification, file type, HTTP domains, organizational unit, department, and role, are subsequently transformed into numerical encoding to facilitate machine learning processing.

We used two types of encoding: Label encoding—to create label-encoded features, such as "PC" (encoded as 0,1,2,3 for own PC, sharedpc, other's pc, and supervisor's pc, respectively). Hot encoding— to extract and populate additional features, for example, HTTP traffic categories (leak, cloud, job) generated features such leak_websites_http_requests and cloud_network_http_requests, and job_websites_http_requests. Similarly, file types (compressed, documents, and executable) generated features such as compressed_files_used, number_of_document_files_used, and number_of_executable_files_used (Le et al., 2020).

Lastly, we ran an Exploratory Data Analysis (EDA) and ensured no missing values or duplicate records were present. Additional EDA steps like Pandas Pearson correlation coefficient and Scipy Zscore were performed to remove correlated features and any statistical outliers introduced by the data processing and extraction steps. In the final stage of EDA, we employ

the Min-Max Scaler from Sklearn to standardize our extracted data. This adjusted all feature values to fall within the [0,1] range, ensuring uniformity in scale and optimizing the dataset for efficient model training and improved performance. Figure 3.5 summarizes the CERT r4.2 feature creation steps.

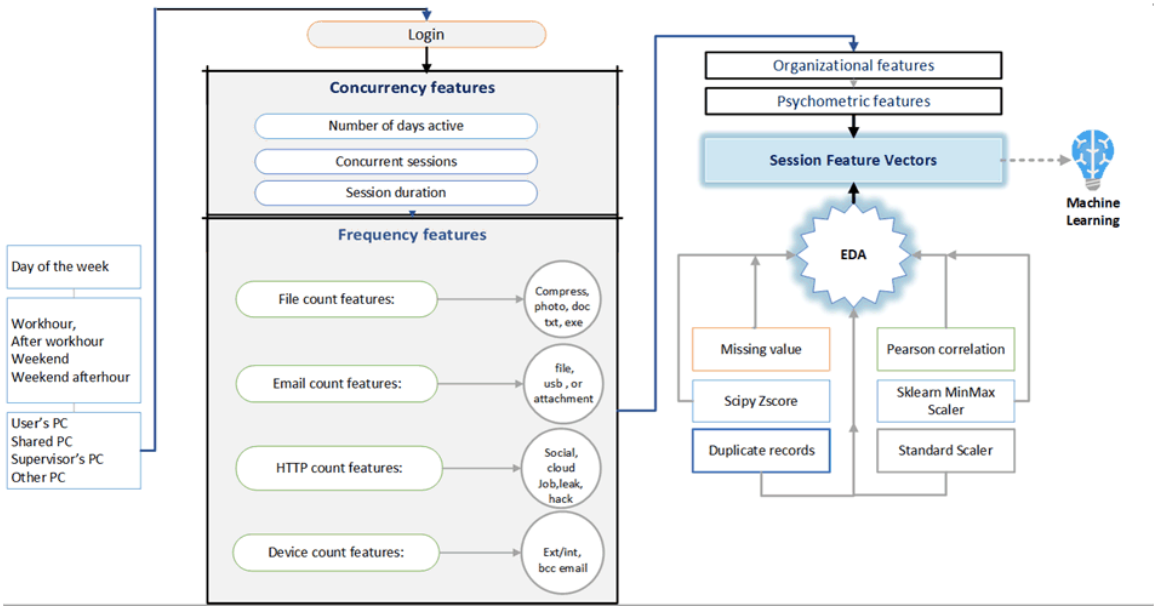


Figure 3.5: *Feature Extraction Process*

After processing the massive amount of records in the CERT r4.2 dataset, we extracted 35 key features and created 470,611 vectors. Of these vectors, 469,497 were labeled as benign, while the remaining vectors were associated with three malicious activity types (69 for scenario 1, 1,013 for scenario 2, and 32 for scenario 3). It is important to note that we employed a binary classification framework, wherein all events were either labeled as either 'Benign' (0) or 'Malicious' (1), irrespective of the distinctions among the specific types of malicious scenarios. Table 3.4 summarizes the final set of features extracted from the CERT r4.2 dataset. "Sessionid" ,"Starttime", and "endtime" will be dropped as they are not needed after we build our User-based Sequencing structure.

Table 3.4: *Extracted Features*

Feature	**Feature**
User_ID	isworkhour
sessionid	isafterhour
starttime	isweekend
endtime	isweekendafterhour
day	number_of_days_session_is_active
pc	session_duration_in_minutes
concurrent_sessions_per_user	user_role
functional_unit	department
is_an_administrator	Openness
Conscientiousness	Extraversion
Agreeableness	Neuroticism
total_number_of_activities	total_number_of_logons
total_number_of_usb_used	total_number_of_files_used
total_compressed_files_used	total_number_of_photo_files_used
total_number_of_document_files_used	total_number_of_ascii_files_used
total_number_of_executable_files_used	total_number_of_emails
total_external_emails	total_external_emails_with_bcc
total_number_of_http_requests	total_social_network_http_requests
total_cloud_network_http_requests	total_job_websites_http_requests
total_leak_websites_http_requests	total_hacking_websites_http_requests

3.5.4 Data Transformation

The importance of transforming tabular data into a sequential format to use in the Transformer stems from the inherent design of the Transformer

models, initially introduced in the context of NLP by Vaswani et al. (2017); the Transformer leverages self-attention mechanisms to process sequential data where the order of the elements is significant. In NLP, this translates to the model's ability to understand the context and the relationships between words in a sentence. Tabular data, on the other hand, is structured differently: It typically consists of rows and columns, where each row represents an instance, and each column represents a feature. However, the relationships between features (columns) are not inherently sequential, and the importance of a feature is not necessarily related to its position in the table (Kolb et al., 2017). Therefore, to harness the power of the Transformer on tabular data, the data must be transformed into a format that mimics a sequence. Our study does this by building User-based Sequencing on top of the extracted dataset vectors.

3.5.4.1 Data Stratification & Split

Since all the CERT datasets have severe imbalanced distribution between benign and malicious sessions, a stratified approach is adopted to ensure the model is trained with a sufficient volume of benign data while also maintaining distinct evaluation sets. Therefore, we used a stratified train/test split approach to ensure a user's session activities are entirely in the training or validation set but not in both (Cervellera & Macciò, 2017).

We first separated the data into two subsets - benign and malicious. For the benign subset, we allocated 70% of the data for model training. The remaining 30% was set aside for further division, with 90% allocated for validation and 10% designated for testing. Furthermore, the 10% testing portion was evenly split into two parts, Benign_Test1 and Benign_Test2, each serving as a distinct set to assess model performance post-training and

validation. The malicious subset underwent a similar partitioning strategy with slight variability. Sixty percent is dedicated to testing the model's ability to identify malicious sessions, and the remaining 40% was combined with the benign data to create a mixed testing data set. Figure 3.6 depicts the data train/test split

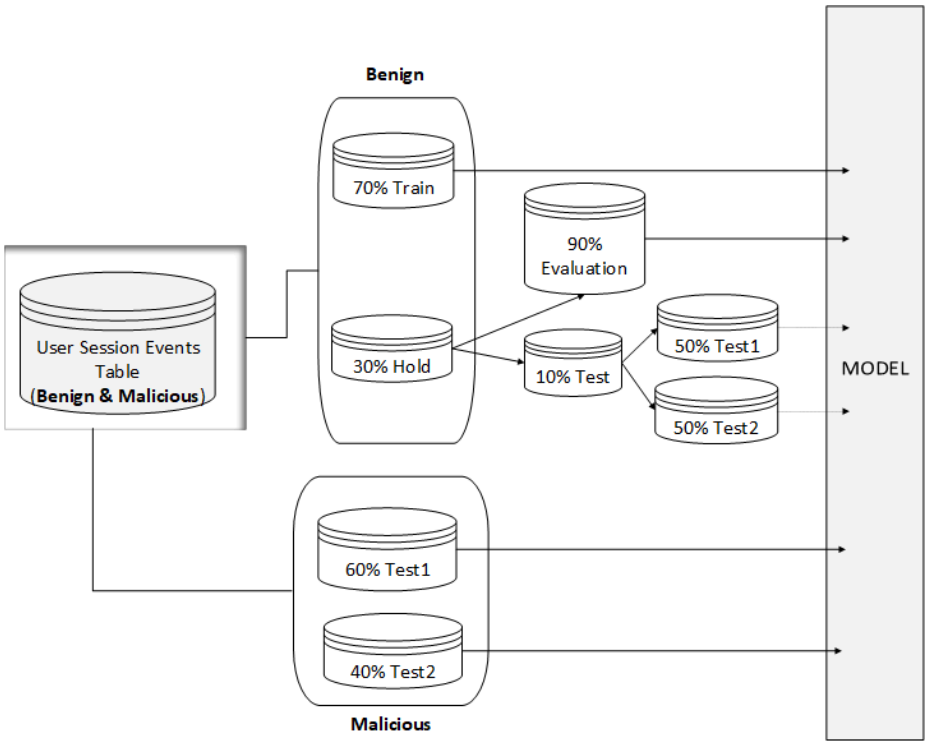


Figure 3.6: *Train/Test split schema*

3.5.4.2 User-Based Sequencing

User-Based Sequencing (UBS) is our novel technique to organize data based on the sequence of actions by users, transforming tabular data into a sequential format suitable for processing by the model. The rationale behind using UBS is that, firstly, we needed to build a separate baseline for each user to capture their temporal and contextual behavior. This allows the model to efficiently detect anomalous behavior by considering

the user's typical activities and preventing minor deviations from being mistakenly flagged as malicious. Secondly, with the help of the adaptive learning capabilities of the Transformer, UBS can enable learning new latent representations of users' behavior as their work patterns change over time (Lee & Kang, 2022). This would greatly reduce the rate of false positives when used as part of an anomaly-based detection system.

For the CERT r4.2, UBS is built as a multi-index Python Dataframe and then converted into a tensor using a dictionary structure identified by the set of users U (1000), number of days D (501), Number of sessions per day S (9), and number of extracted features F (35), such that: For a single user u, the tensor that represents the [501,9,35] dimensions can be denoted as T_u. Hence, $T_u[i][j][k]$ represents the kth feature of the jth session on the ith day for user u. The entire user data dictionary can then be expressed as: user_data: U -> T. Where T is the tensor defining all possible tensors of shape [501,9,35]. It is important to note that S is set based on the maximum number of sessions extracted from the CERT dataset. To allow an additional buffer for future growth; we set S to 9. However, this can easily be increased if the number of sessions per user grows beyond that. Figure 3.7 depicts our UBS structure and Table 3.5 summarizes the User-Based Sequencing Pseudo-code.

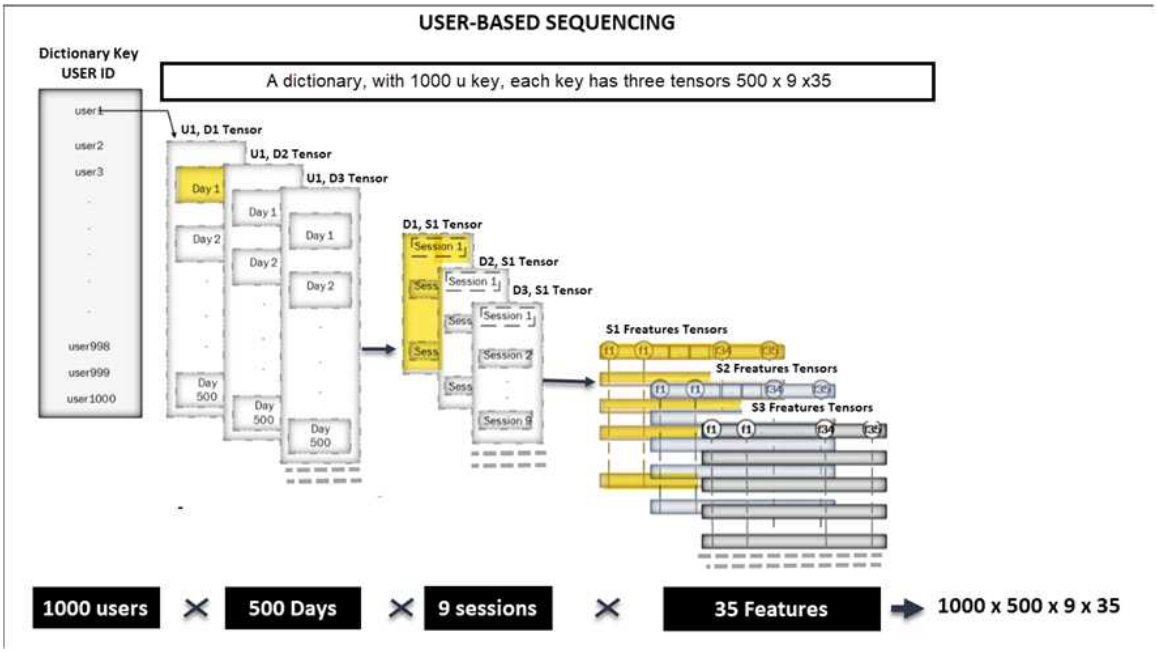


Figure 3.7: *User-Based Sequencing pictorial representation*

Table 3.5: *Pseudo-code steps to build User-Based Sequencing structure*

- **Step**
1. Initialize userDayCombination from the original tabular data frame. 2. Initialize an empty list (transformerRecords).
3. **For** each (user, day) in UserDayCombinations: - Initialize a new record. - Append features to the new record. - Determine the maximum number of sessions for the user.
if the user has no session for the day: - Create a placeholder record and populate it with zeros. **else**: - Add a new record to transformedRecords.
4. Convert transformedRecords into a multi-index dataframe. 5. Convert the multi-index dataframe into a dictionary of tensors:
- **For** each user in the dataframe: - Use the user ID as the key. - Convert the user data into a tensor of shape $500 \times 9 \times 35$.

3.5.5 Model

Anomaly detection aims to identify events that deviate significantly from the normal behavior. Our proposed Transformer model is trained exclusively on benign data to learn what constitutes "normal" behavior and detect anomalies when they arise. This is achieved through reconstruction error, where the model tries to reconstruct its input and flag significant differences as anomalies (Zhao & Sun, 2023). This is generally measured by how large the reconstruction error is, determined after setting a threshold as a cut-off point. The threshold is commonly determined through statistical methods using cross-validation or domain-specific knowledge; however, as we mentioned in Section 3.4, we use LOF, OCSVM, and iForest to identify anomalies in the reconstruction errors.

3.5.6 Baseline

To compare the performance of our Transformer, we also build a baseline model using the Autoencoder. This serves as a dual purpose. Firstly, it enables us to compare the performance between the Autoencoder using tabular data and the Transformer using UBS. Secondly, it allows us to also run the Autoencoder on UBS data and evaluate how our UBS enhance or degrade the Autoencoder performance. This is needed because we wanted to evaluate our second research hypothesis by seeking to understand the impact of a different data representation on insider threat detection.

In addition to our primary baseline autoencoder model, we also used the standalone LOF, OCSVM, and iForest models to evaluate their performance against both the Transformer and Autoencoders. Furthermore, to rigorously assess the performance differences between the Transformer and autoen-

coder models, we use the McNemar statistical hypothesis testing to compare their performance. Finally, to gain insights into the decision-making process of our model, we employed SHapley Additive exPlanations (SHAP). However, it is important to note that our application of SHAP primarily focuses on evaluating the decisions made by the LOF, OCSVM, and IForest based on the reconstruction errors previously generated by the Transformer. As such, while SHAP offers valuable explanations, its utility is explicitly tied to analyzing these anomaly detection algorithms' decisions rather than providing a direct interpretation of the Transformer's internal decision. Figure 3.8 presents an end-to-end workflow of our insider threat detection system.

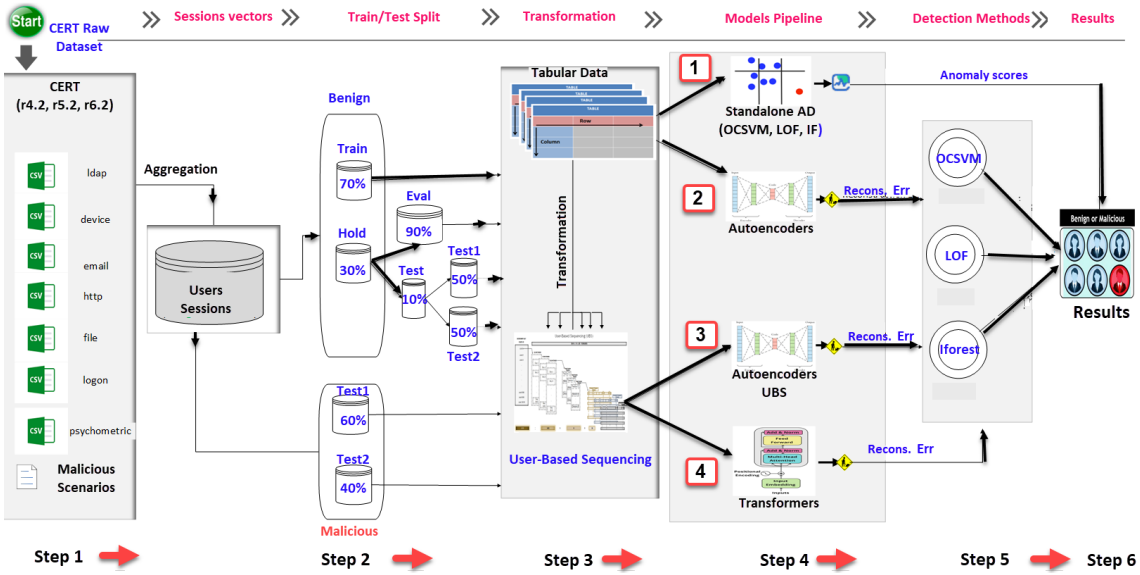


Figure 3.8: *Insider threat Model Workflow*

3.5.7 Transformer Encoders

This research primarily uses the "Encoder" part of the vanilla transformer architecture. The rationale behind this decision is that we only aim to train the Transformer on what is "normal" and calculate the reconstruction errors from subsequent input. However, we must still change some Encoder

settings to accommodate our data. For instance, we added an initial linear layer as an embedding layer to project our User-Based Sequencing input dimensions correctly. In addition, Since the self-attention mechanism in the Transformer is inherently order-agnostic, positional encoding are added to the input embedding to provide the model with information about the order of the features in our UBS structure. Furthermore, we added a dropout layer after the Positional Encoding to serve as a regularization technique, which, during our testing, gave a better result. The pictorial representation of our Transformer encoders layer is depicted in Figure 3.9. All encoders have the same structure.

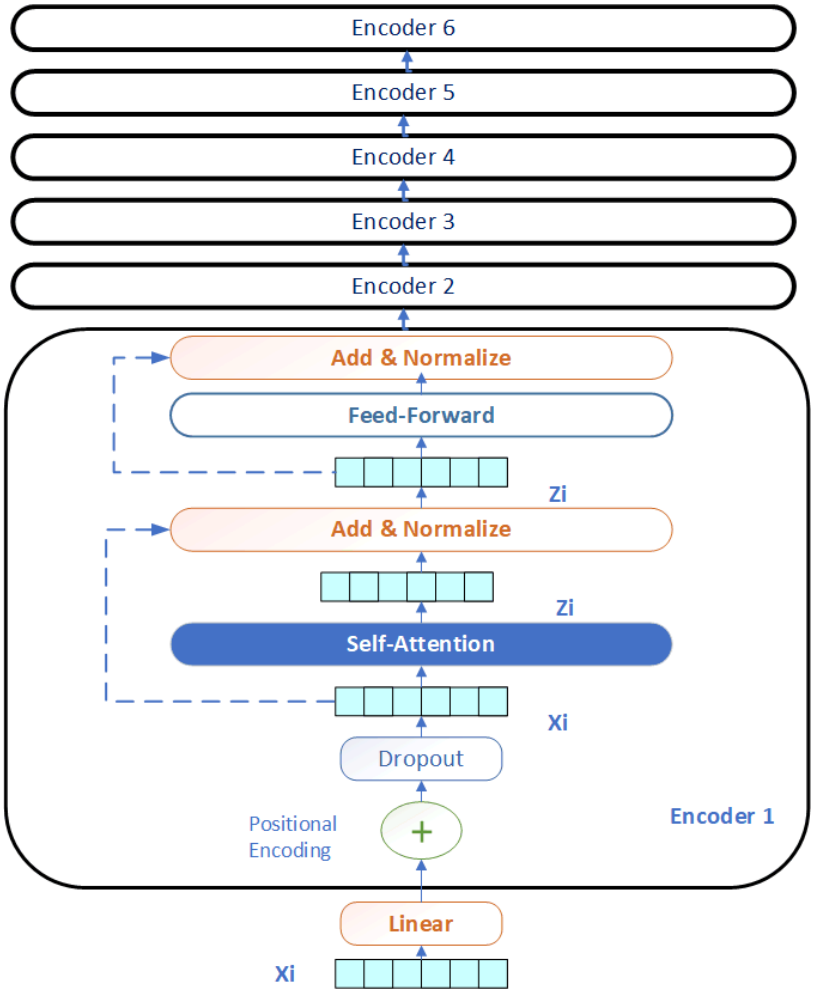


Figure 3.9: *Encoder Pictorial Layer Representation*

3.5.8 Hyper-parameters

The Transformer relies on hyper-parameters that require precise tuning for optimal performance. Adjusting these hyperparameters to align with the data's distinct attributes can significantly improve the model's effectiveness. This involves running the algorithm multiple times with different hyperparameter settings to determine the combination that yields the best performance on a validation set. In order to ensure our model training is perfectly in sync with our UBS structure, we use a batch size of "1", referred to this as User-Based Batching (UBB). This allows the model to learn from the entire user space at once. Furthermore, We ran multiple epochs, 50, 100, 200, and 500, but found that 57 epochs were optimal for our needs. We used Mean Squared Error (MSELoss) as our Loss function and Adaptive Moment Estimation (ADAM) (Kingma & Ba, 2014) as our optimizer because it efficiently adjusts learning rates and uses fewer resources to converge. In addition, We fine-tuned the internal Transformer neural network's settings, such as the number of layers, nodes, dropout, and learning rate, by testing various combinations of these hyperparameters using a Cartesian product to ensure no repetition in these combinations (Agesen, 1995). Ultimately, the best set of hyperparameters we experimented with yielded a model with 5,387,579 trainable parameters. Table 3.6 outlines the hyperparameters we fine-tuned.

Table 3.6: *Transformer Encoder hyper-parameters*

Hyper-parameters	**Default**	**Adjusted**
num_encoder_layers: Number of layers	6	6
num_heads	8	8
d_model: Overall dimensionality of the model	512	320
DIM_feedforward	2048	512
input_dimension	512	315
batch	False	1
LR	False	0.00001
epochs	False	200, 57
dropout	0.1	0.1
Loss Function	False	MSELoss
Optimizer	False	Adam

3.5.9 Detection Algorithms

In our experiment, we needed a mechanism to set the anomaly threshold to measure the deviation from the reconstruction errors produced by the transformer and other baseline models. The standard way to set the threshold is to use statistical methods like the mean/standard deviation or MAD. However, these methods are prone to outliers and may heavily be influenced by the data distribution.

Inspired by the work of (Haidar & Gaber, 2018), (Emmott et al., 2015), and (Bartoszewski, 2022), we decided to try a novel technique of using LOF, OCSVM, and iForest to detect outliers in our reconstruction errors.

LOF introduced by Breunig et al. (2000) is an anomaly detection algorithm that works by measuring the local density deviation of a given data point with respect to its neighbors. Points that have a substantially lower density than their neighbors are considered outliers. iForest introduced by F. T. Liu et al. (2008) is an anomaly detection method based on decision trees. It works by randomly selecting a feature and a split value between the maximum

and minimum values of the selected feature. On the other hand, OCSVM is a special variant of regular Support Vector Machine (SVM). It attempts to find a hyperplane in a high-dimensional space that best separates the data points from the origin, thereby maximizing the distance from the hyperplane to the origin. It aims to create this hyperplane as far as possible from the anomalous points while containing the normal data points. Anomalies in OCSVM become the center of the coordinate system (Schölkopf et al., 2001).

We did not find any cases in the literature using LOF, OCSVM, and iForest to evaluate the reconstruction errors. We believe this is a suitable choice for the problem and would be a novel contribution to the field regardless of the results. However, after implementing these algorithms, the results showed the performance is excellent, validating our intuition of using them. More details about the result of this experiment are discussed in Chapter 4.

3.6 Conclusion

In this chapter, we outlined the methodology for detecting insider threats using a binary anomaly detection strategy. Our approach involves training a transformer on normal data and then using the construction error to evaluate if the test data is normal or malicious. This is all made possible because of our novel User-Based Sequencing structure to convert the CERT tabular data into a format suitable for the Transformer architecture. Moreover, Instead of using a statistical procedure like the mean and standard deviation or MAD to set the anomaly threshold manually, we used machine learning algorithms, including OCSV, LOF, and iForest, to evaluate the reconstruction errors. This methodological approach addresses the research questions and hypotheses outlined in Chapter 1 and repeated below:

RQ1: How do Transformer encoders compare to Autoencoders in terms

of accuracy, precision, and recall when detecting malicious insider threats?

RQ2: How does user-based sequencing improve the accuracy of insider threat detection?

RQ3: How can a model based on transformer encoders be optimized to enhance both speed and detection rate across various versions of the CERT insider threat datasets?

H1: Transformer encoders can capture long-range dependencies better than Autoencoders and accurately detect malicious insider threats.

H2: User-based sequencing enables the implementation of a Transformer Encoder on tabular data to improve the detection of insider threats.

H3: Using feature optimization will improve the speed and detection rate across various CERT dataset versions.

Chapter 4: Results and Analysis

4.1 Introduction

This chapter presents the results of our experiment using the methodology outlined in Chapter 3. We aim to evaluate these results guided by the research questions and hypotheses introduced in Chapter 1. The chapter is organized into several sections. The first section introduces the Anomaly Detection (AD) approach, dataset statistics, and performance metrics used to evaluate the experiment results. Section 2 presents the experiment results using our User-Based Sequencing (UBS) as well as tabular data. The third section analyzes and discusses the consolidated results. The chapter is concluded by evaluating our hypothesis using the McNemar test and interpreting the results using SHapley Additive exPlanations (SHAP).

4.2 Anomaly Detection Framework

This research concludes that detecting insiders threats can be significantly enhanced when employing an AD framework to scrutinize each event in the broader context of the user's overall activities. We adopted a similar unsupervised anomaly detection approach as discussed by Chandola et al. (2009). The strength of unsupervised anomaly detection lies in its reliable technique, where the model is only trained on "normal" data points (F. T. Liu et al., 2008). However, our approach for training and evaluating the result based on reconstruction error is more advanced and effective than what was outlined by Chandola et al. (2009). We used a novel user-based sequencing structure to enable our models to accurately learn and recon-

struct data points. In addition, we used Machine Learning (ML) algorithms instead of relying on a predefined threshold to determine if the difference in reconstruction errors can be flagged as an anomaly. Figure 4.1 illustrates our anomaly detection workflow.

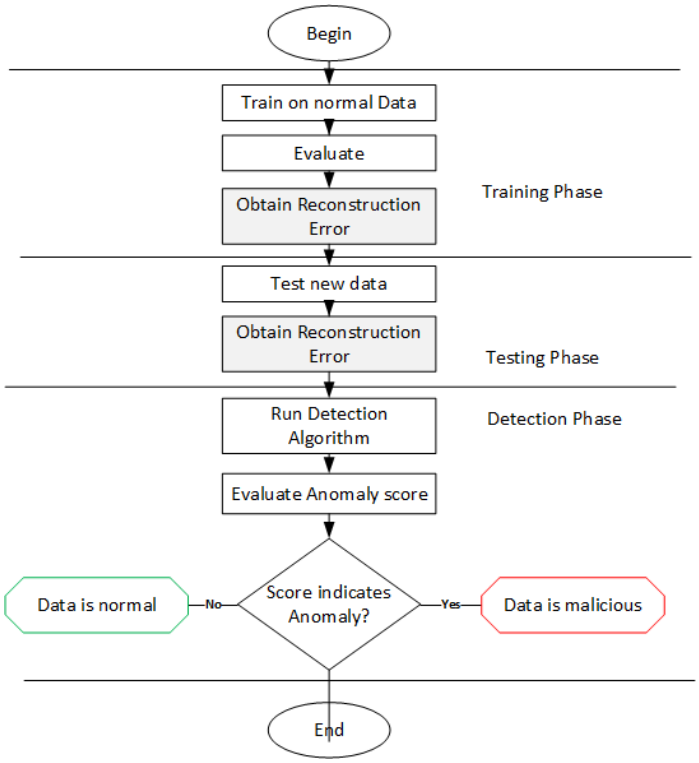


Figure 4.1: *Anomaly detection workflow*

4.3 Dataset Statistics & Distribution

The dataset used in this study, CERT r4.2, does not explicitly categorize individual user events as either malicious or benign. However, out of the 1000 users tracked for 17 months, 70 were identified as malicious. The sequence of events carried out by those malicious users is available in the "insiders.csv" file and is categorized into three scenarios in the "scenarios.txt" file: Exfiltration (scenario 1), Theft (scenario 2), and Sabotage (scenario 3). A malicious user can have a few dozen events over multiple

days or a few hundred events over a more extended period. In essence, there is no concept in the CERT dataset of one user session being malicious.

To demonstrate the complexity of insider threats, we randomly sampled three malicious users from the CERT r4.2 dataset to investigate the temporal evolution of their malicious behaviors. Figure 4.2 summarizes their activities. For instance, user "AAM0658" first session started on "10/23/2010" at 1:34:19 and ended on "10/23/2010" at 06:28:17. The second session for the same user started on "10/29/2010" at 12:06:47 and ended on "10/29/2010" at 5:23:28. By only considering the first session for user "AAM0658", it is not definitive to conclude the user is malicious. This is correct because the user is only identified as malicious after considering both sessions. The same scenario applies to users PNL0301 and BSS0369, whose malicious activities span 50 and one day, respectively.

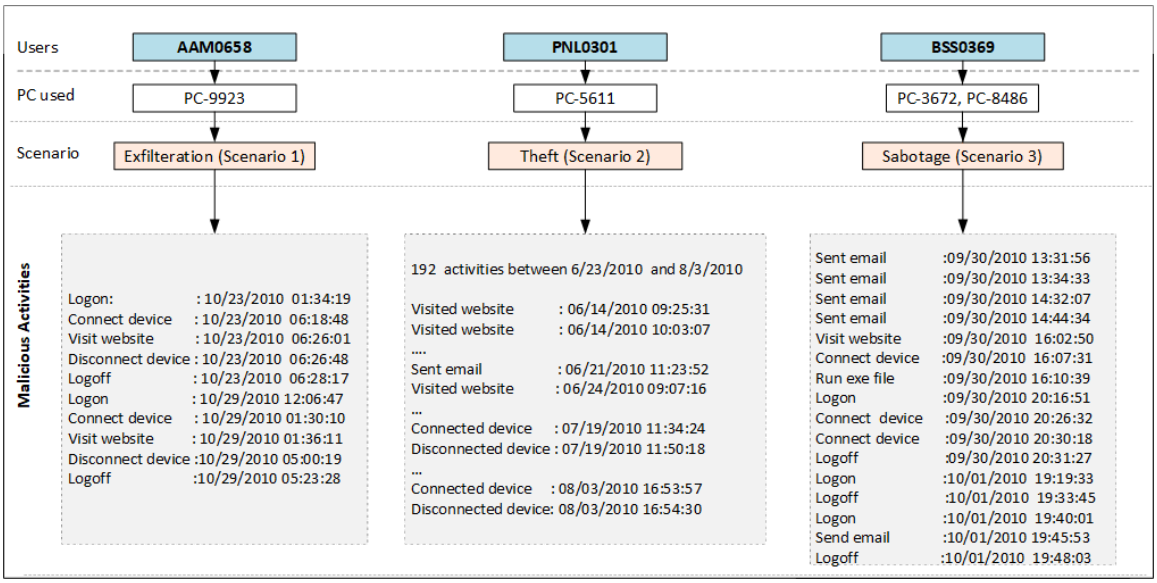


Figure 4.2: *Malicious Insider Activities sample from the CERT r4.2*

In Chapter 1, we formulated our research hypotheses to address insider threats regardless of the type of malicious activities they performed. Hence, we use a binary classification approach to distinguish between malicious

and benign users. To achieve this, we have categorized all attack scenarios within the CERT datasets into a binary label of '1' to denote malicious users. Conversely, non-malicious activities are grouped under 'Benign user activities' and assigned a binary label of '0'. In Chapter 3, we addressed the significant challenge posed by the imbalance in the CERT dataset and the preponderance of normal activities over malicious insider scenarios. Figure 4.3 [Fig: A] depicts this disparity, highlighting the skewed distribution of insider threats in the CERT r4.2 dataset with merely 69 instances of exfiltration, 1,013 instances of theft, and 32 instances of sabotage recorded. After aggregating all attack scenarios under one label— 'binary 1', the total malicious activities account for only *0.2%* of total user activities as depicted in Figure4.3 [Fig: B].

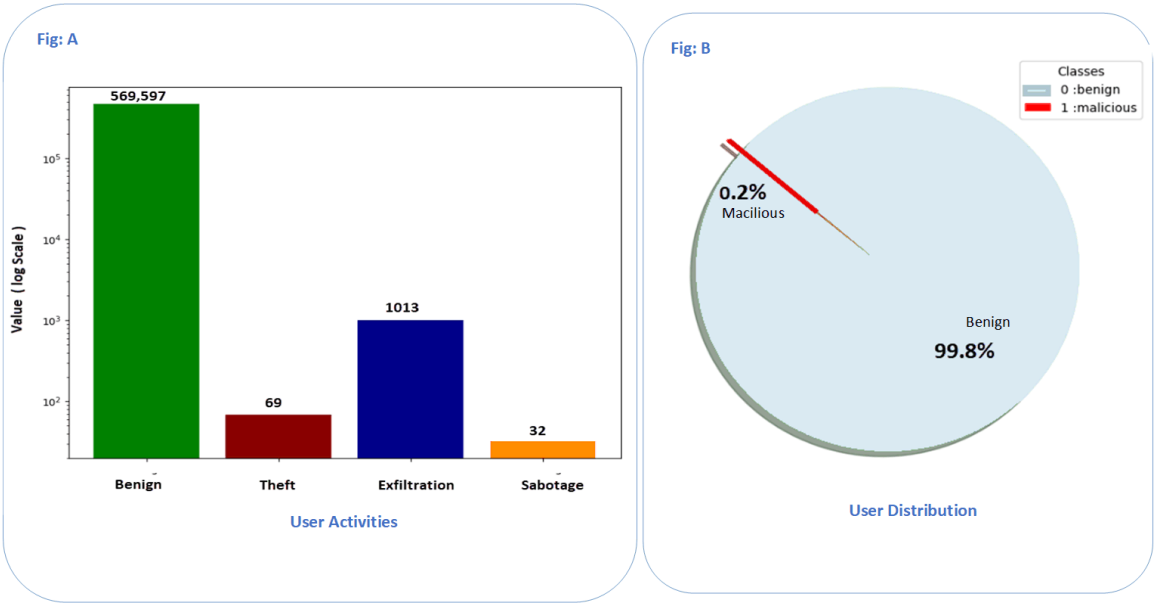


Figure 4.3: *Insider Attacks Distribution - CERT r4.2*

As mentioned in the previous section, among the 1,000 users initially tracked in the CERT r4.2 dataset, 70 were identified as malicious. Figure 4.4 presents the average daily activities of all users, and Figure 4.5 presents

the count of benign and malicious activities performed by the malicious users.

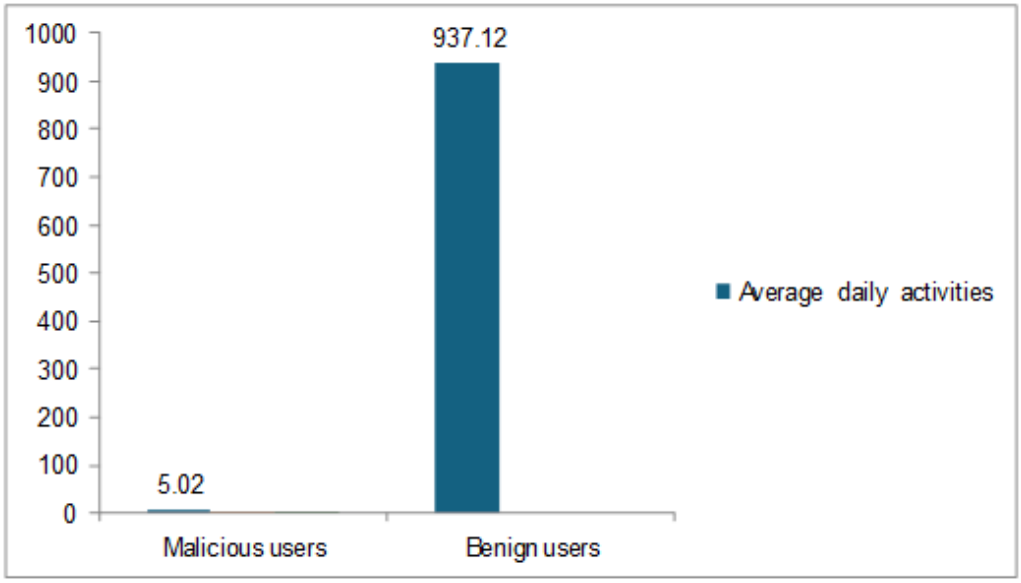


Figure 4.4: *Average Daily User Activities - CERT r4.2*

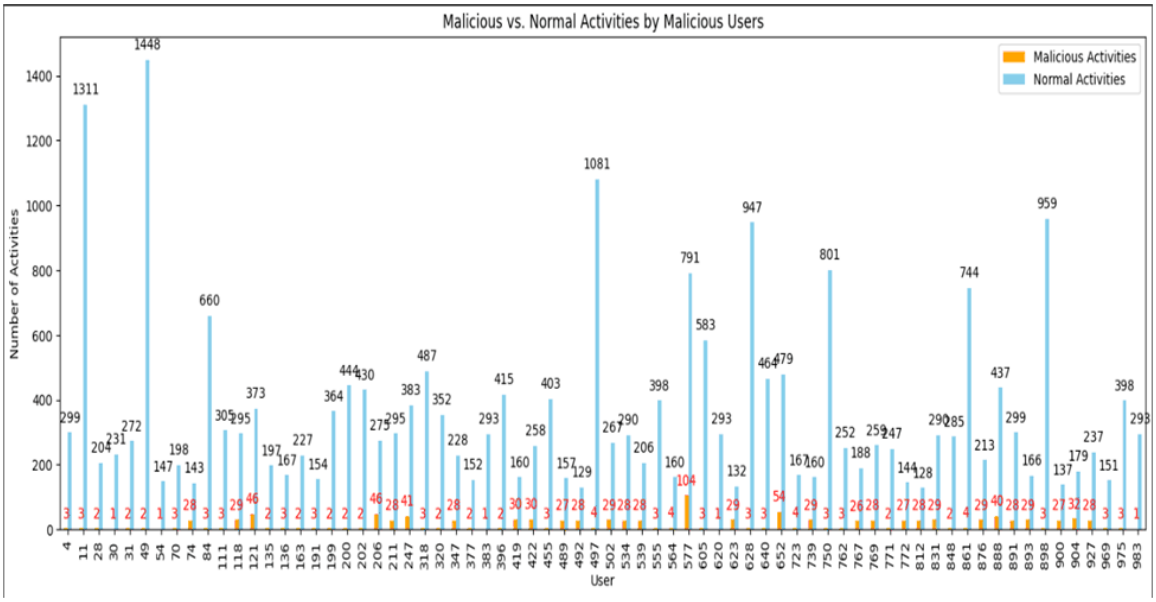


Figure 4.5: *Malicious User Activities Counts - CERT r4.2*

In addition, Table 4.1 presents an overview of the activities performed by malicious and benign users in the CERT r4.2. The table includes a range

of metrics, such as the distribution of activities across different periods and specific actions like USB and HTTP usage. The significant difference in activity levels between malicious and benign users is apparent, as benign users display notably higher activity counts across all metrics. This disparity emphasizes the complex behavior of malicious users who may engage in fewer activities but could pose a greater risk and impact.

Table 4.1: *User Activities Statistics Summary - CERT r4.2*

Metric	Malicious	Benign
Activities during work hours only	376	224,763
Activities during after hours only	118	33,976
Activities spanning work and after hours	604	198,957
Activities during weekends	16	12,034
Activities involve USB	1,007	75,680
Activities involve HTTP	1,044	416,489
Activities involve files	685	58,113
No. of activities in most busy day	13	1,503
No. of activities in least busy day	1	4
Total malicious activities	1,114	469,497

For the final phase of the dataset analysis, we needed to determine the maximum number of concurrent sessions a user could have within a day. This particular statistic plays a crucial role in building our UBS structure as it defines the — "maximum session per day"— rule, one of the key metrics we use to construct our UBS structure. From Figure 4.6, we concluded that the Maximum number of concurrent sessions is 9, which we used as the current threshold for building our UBS structure. However, as explained in Chapter 3, this threshold can easily be adjusted upwards or downwards to accommodate potential future session shifts.

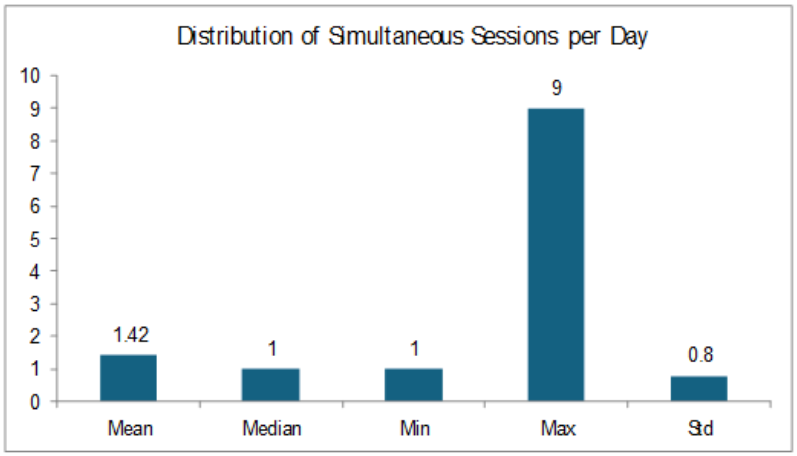


Figure 4.6: *Statistical-Distribution of Simultaneous Sessions Per Day*

4.4 Classification Metrics

Classification metrics are crucial for quantifying and evaluating the performance of various models and algorithms. They provide a more comprehensive assessment of the model's effectiveness beyond just its overall accuracy. These metrics are especially vital in areas like threat detection, where the cost of overlooking a threat can be extremely high. The following sections will introduce the basic evaluation metrics typically used to assess classification and anomaly detection models.

In machine learning, the confusion matrix is crucial for assessing algorithm performance. This tabular representation compares predicted values with actual outcomes, categorizing each instance as True Positive, False Positive, True Negative, or False Negative (Hossin & Sulaiman, 2015). Understanding these categories is vital for determining the algorithm's accuracy and pinpointing areas for improvement. The visual representation in Figure 4.7 illustrates the various elements of the Confusion Matrix.

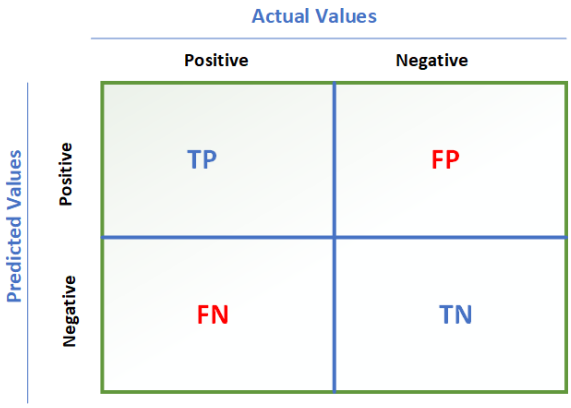


Figure 4.7: *Confusion Matrix*

- **True Positive (TP):** This metric denotes instances where the model accurately predicts positive outcomes, thereby indicating its effectiveness in correctly identifying the presence of the specific conditions or characteristics being analyzed.

- **False Positive (FP):** This occurs when negative samples are mistakenly classified as positive. It emphasizes the model's tendency to overpredict the presence of a condition, potentially leading to false alarms.

- **True Negative (TN):** This refers to cases where the model correctly identifies negative samples, indicating the model's ability to recognize the absence of the condition being tested.

- **False Negative (FN):** This metric refers to instances where the model incorrectly classifies positive samples as negative. It indicates a failure to detect the presence of the condition, which can be particularly concerning.

4.5 Metrics Definition

This study, like previous studies (Le & Zincir-Heywood, 2021a; Le et al., 2020; Wang & El Saddik, 2023) uses a wide array of metrics to evaluate and compare the models' performance. These include Accuracy (A), Precision (P), Recall (R), and F-1 score (F1). In addition, we use anomaly scores produced by the three detection algorithms— One-Class Support Vector Machine (OCSVM), Local Outlier Factory (LOF), and Isolation Forest (IFOREST)— to drive metrics such as Receiver Operating Characteristic Curve (ROC), Anomaly Score Distribution, False Negative Rate (FNR), and False Positive Rate (FPR).

Accuracy represents the ratio of correctly predicted observations to the total observations. It provides a straightforward measure of a model's overall effectiveness but can be deceptive when dealing with imbalanced datasets. The following equation expresses accuracy:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

Precision evaluates the fraction of true positive events among all instances predicted to be positive. A high precision score reflects a low number of false positives. The Precision can be calculated using the following formula:

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

Recall, also known as Sensitivity or Detection Rate (DR), quantifies the percentage of true positive instances that the model accurately detects (Hossin & Sulaiman, 2015). The formula for recall is given by:

$$Recall\ (Sensitivity) = \frac{TP}{TP + FN} \tag{4.3}$$

The **F1 Score** is the harmonic mean of precision and recall (Hossin & Sulaiman, 2015). It is particularly useful when a balance between precision and recall is needed, offering a single metric to evaluate their combined effectiveness.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{4.4}$$

The **AUC-ROC** (Area Under the Curve - Receiver Operating Characteristic curve) is a performance metric used in classification problems to assess model performance at different threshold settings. The ROC is a probability curve, and the AUC measures the overall ability of the model to differentiate between classes. It is calculated by determining the area under the ROC curve, which plots the True Positive Rate (TPR) on the y-axis and the False Positive Rate (FPR) on the x-axis across various thresholds (Hossin & Sulaiman, 2015).

False Positive Rate (FPR) quantifies the fraction of actual negative instances that are incorrectly labeled as positive. For example, it represents the percentage of benign users mistakenly identified as malicious. It is calculated as follows:

$$False\ Positive\ Rate\ (FPR) = \frac{FP}{FP + TN} \tag{4.5}$$

The **False Negative Rate** (FNR), also known as the Miss Rate, measures the frequency at which actual positive instances are incorrectly classified as negative. In other words, it indicates the proportion of true positives that the model fails to identify, classifying them as negatives instead. The FNR

is computed using the following formula:

$$Specificity = \frac{FN}{FN + TP} \tag{4.6}$$

In this research, we emphasize recall, FPR, and FNR to measure the model's effectiveness in detecting malicious insider threats. A high FPR can exhaust security analysts by generating too many false alarms, potentially leading to alarm fatigue. Conversely, a high FNR can result in missing actual threats. Striking a balance between minimizing both FPR and FNR is important to ensure the effectiveness and reliability of our anomaly detection approach.

4.6 Experimental Setup

This section discusses the setup and design of our experiment. As discussed in Chapter 3, we split the CERT r4.2 data into 70/30 training and evaluation. However, given that our anomaly approach relies on identifying malicious users based on a sequence of events, we decided that analyzing the entire user activity is necessary to ascertain if a user is malicious. This is because more than a single session is required to make such a determination. Therefore, we split the entire user space in the CERT r4.2 using our 70/30 split into 700 users for training, 270 for evaluation to calculate the reconstruction error, and 30 for testing. Our stratified splitting approach ensures that all users across these sets are distinct. However, since more than 30 users are required to test our model thoroughly, we included users from CERT r5.2 and r6.2 in our test sets. This decision proved valuable as it allowed our model to be tested on multiple datasets generated at different times and with some variations from the training

dataset.

4.6.1 Test Sets

As evident from the large number of benign versus malicious users in the CERT dataset—r4.2: 1000 versus 70, r5.2: 1901 versus 99, and r6.2: 3995 versus five users, we needed a way to construct test sets that are regardless of the CERT dataset version used. Table 4.2 summarizes the test we built to address this problem.

Table 4.2: *Model Test-Sets Summary*

Testset	Version	Benign Users	Malicious Users	Total Users
1	r4.2	30	70	**100**
2	r5.2	60	99	**159**
3	r6.2	120	5	**125**
4	r4.2, r5.2, r6.2	210	174	**384**

4.6.2 Model setup

In Chapter 3, we outlined the methodology for validating the first and second Hypotheses using two models: the Transformer and Autoencoder. While the Autoencoder is well-suited for tabular data without requiring any transformation, the Transformer, on the other hand, requires a sequential format. As mentioned before, we developed a novel UBS structure to address the transformer need, but we also preserved the tabular data to use elsewhere. Henceforth, our model testing comprises three combinations: First, we evaluate the Transformer against the Autoencoder (using Tabular data), referred to as AutoTAB - to validate the first hypothesis. Second, we evaluate the Transformer against the Autoencoder (using UBS data),

referred to as AutoUBS - to validate the second hypothesis. Finally, we compared AutoTAB versus AutoUBS to evaluate our third hypothesis.

4.7 Detection Algorithms

The primary objective of anomaly detection is to discern instances that markedly diverge from established patterns of normal behavior. To this end, our models, including the Transformer and baseline, are exclusively trained on benign data. This training regimen equips the models to generate what is known as 'reconstruction errors.' These errors effectively mirror the models' internal data representation of what they have learned. Consequently, any significant differences in the reconstruction errors between training and testing—exceeding a predefined threshold—can be used to flag the record as anomalous (Zhao & Sun, 2023). Statistically, this is measured by how large the difference is from a predetermined threshold, and the threshold is commonly determined through methods like the mean, standard deviation, and MAD or using cross-validation or domain-specific knowledge. However, these manual methods are prone to outliers and may be heavily influenced by the data distribution. Therefore, as mentioned in Section 3.4, we used ML algorithms like LOF, OCSVM, and IFOREST to evaluate the reconstruction errors and identify anomalies.

One of the key strengths of using ML detection algorithms is the valuable information provided by the anomaly scores they generate. These scores, which range from 0 to 1, represent the probability of a record being anomalous. Not only does this enable us to quickly generate ROC curves to assess the sensitivity and specificity of each model, but it also allows us to calculate additional performance metrics such as FPR and FNR. However, to ensure the optimal performance of our ML detection algorithms, we needed

to fine-tune multiple hyperparameters to achieve uniform test results across all test sets. Table 4.3 summarizes the most effective hyperparameters we found that work best in all test sets.

Table 4.3: *Detection Algorithms Hyperparameters*

Algorithm	Parameter	Value
OCSVM	'cache_size'	200
	'coef0'	0.0
	'degree'	3
	'gamma'	'auto'
	'kernel'	'rbf'
	'max_iter'	-1
	'nu'	0.05
	'shrinking'	True
	'tol'	0.001
LOF	'algorithm'	'auto'
	'contamination'	0.05
	'leaf_size'	30
	'metric'	'minkowski'
	'n_neighbors'	20
	'novelty'	True
	'p'	2
IFOREST	'bootstrap'	False
	'contamination'	0.05
	'max_features'	1.0
	'max_samples'	'auto'
	'n_estimators'	100
	'warm_start'	False

4.8 Model Training & Evaluation

This section provides insights into the model training and evaluation performance and sets the stage for a detailed discussion of the performance metrics in the next section.

The loss trajectory during training and evaluation indicates how well a model learns from the data. In our case, since we employ two different data structures —UBS and Tabular—the loss trajectory between the Transformer and Autoencoder should give us a pretty good idea of how well our UBS

strategy plays as a precursor to improving model performance when we run the test data.

Transformer Losses: The Transformer's outstanding aptitude for learning from UBS is demonstrated through the noticeable convergence of its training and evaluation loss, achieved in less than 50 epochs. The Transformer, well-suited for handling sequential data, quickly took advantage of the built-in temporal user behaviors embedded in our UBS structure. Figure 4.8 depicts the Transformer's training and evaluation losses trajectory.

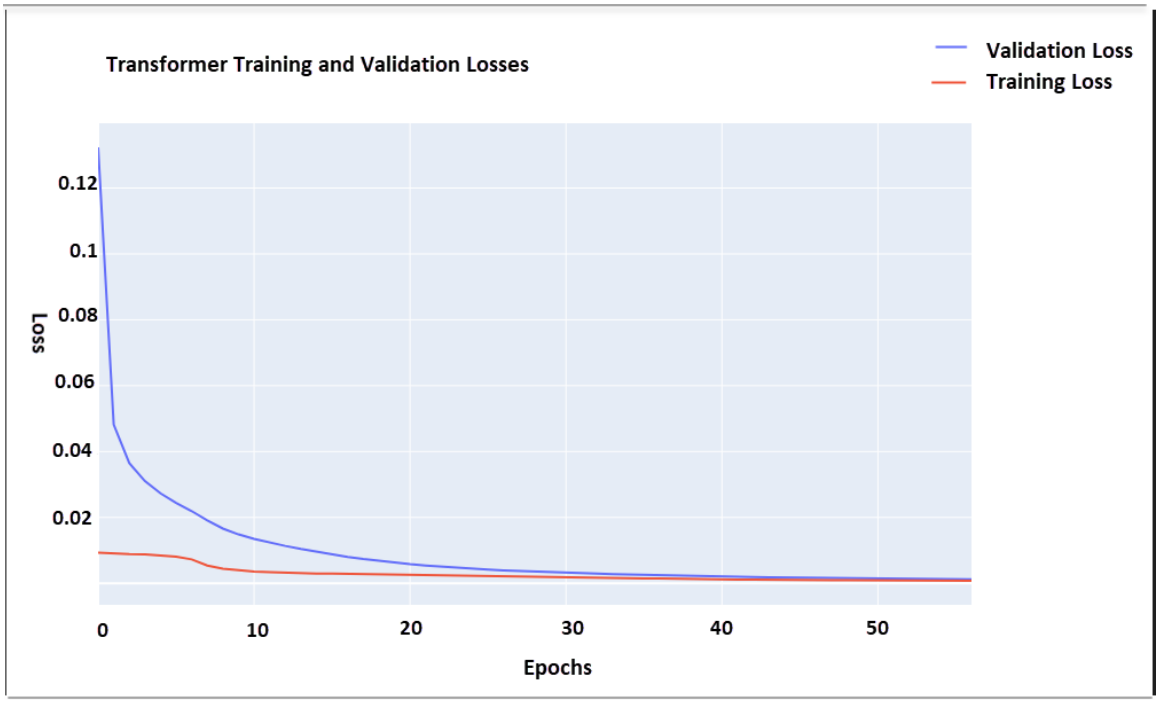


Figure 4.8: *Transformers Losses*

Autoencoder Losses: The stark differences in loss trajectory between the two Autoencoder models— AutoTAB & AutoUBS— underscore UBS's important role beyond enabling the Transformer modeling. Since Autoencoders are typically used for feature extraction and dimensionality reduction in unstructured or semi-structured data, they were ineffective when used against our tabular data, which lacks user temporal and sequential patterns

compared to our UBS structure. This limitation is apparent in the AutoTAB constant loss trajectory that fails to converge as the epoch increases. In contrast, the Autoencoder using UBS exhibits notable progress, as evidenced by decreasing training and evaluation losses. Figure 4.9 depicts the loss comparison between the AutoTAB and AutoUBS models.

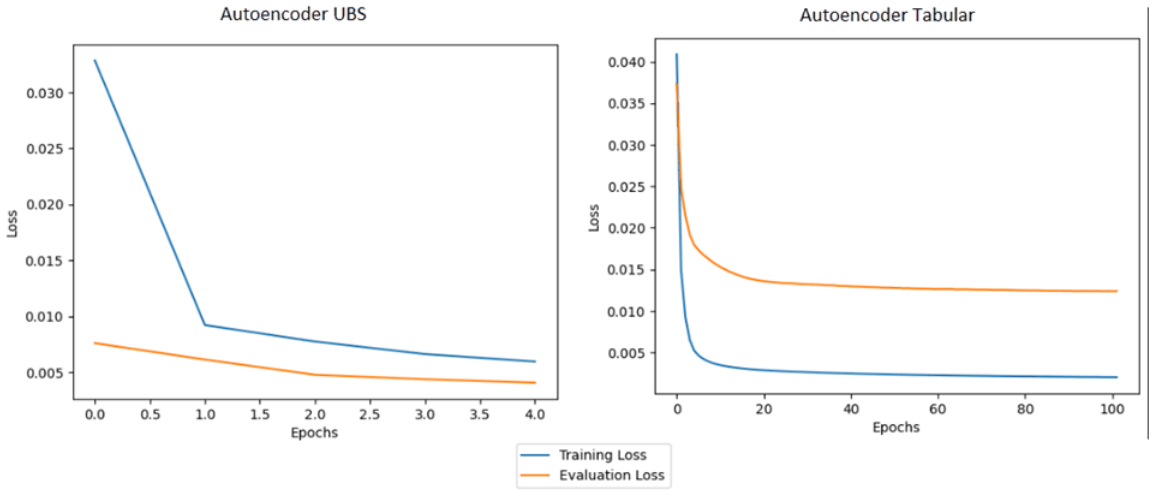


Figure 4.9: *Losses Comparison (AutoUBS vs. AutoTAB)*

In summary, our UBS data transformation has proven to be highly effective in improving the models' ability to learn from the embedded temporal and sequential patterns, regardless of the specific model employed. This is further supported by the results obtained from each model's confusion matrices, ROC curves, and anomaly score distribution. As demonstrated in Figure 4.10, we clearly see the differences in performance between the Transformer, Autoencoder-Tabular, and Autoencoder-UBS models. Figure 4.10 is just one example from our OCSVM detection algorithm where subfigures (A, B, C), (D, E, F), and (G, H, I) represent the Transformer CM, distribution of Anomaly Scores, and ROC for the Transformer, Autoencoder-TAB, and Autoencoder-UBS, respectively. Results from other detection algorithms are

included in Appendix A.

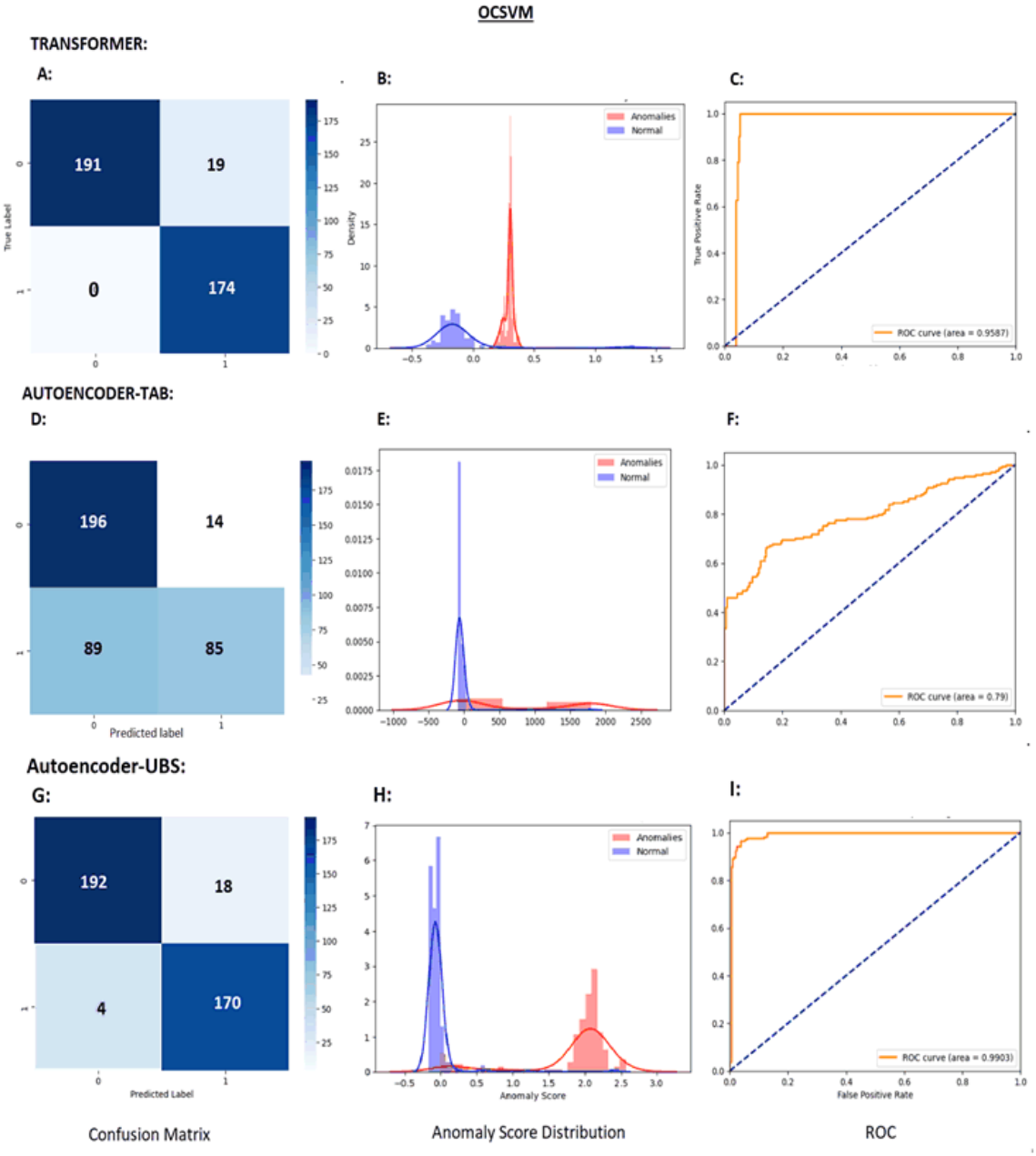


Figure 4.10: *Transformer vs. Autoencoders CM, Anomaly Score, & ROC*

4.9 Transformer Versus Autoencoder Results

This section analyzes and compares the performance between the Transformer, AutoTab, and AutoUBS using Test-1, Test-2, and Test-3 as described

in Table 4.2.

4.9.1 Transformer Versus AutoTAB

4.9.1.1 Test 1: CERT r4.2 Dataset

For the Transformer, all three detection methods—OCSVM, LOF, and iFOREST—demonstrated exemplary performance with an accuracy of 99%, precision of 98.59%, a recall of 100%, F1 score of 99.29%, and a perfect AUROC score of 100%. The FNR remained at 0% across all detection algorithms, indicating an excellent ability to identify true positives. The FPR was consistent at 3.33%, suggesting a minimal rate of falsely classifying benign users as malicious. In contrast, the AutoTAB model exhibits notable variance across all detection methods. Nonetheless, the iFOREST scores better in terms of accuracy at 79%, with a high precision of 96.23% and a moderate recall of 72.86%. However, the Transformer by far exceeded the AutoTab in all metrics across all detection methods. Table 4.4 depicts the stark performance difference between Transformer and AutoTAB using Test-1 based on the CERT r4.2 dataset.

Table 4.4: *Test-1: Transformer vs. AutoTAB Performance Metrics*

Model	Detection Method	A	P	R	F1	AUC	FNR	FPR
Trans	OCSVM	**0.9900**	**0.9859**	**1.0000**	**0.9929**	**1.0000**	**0.0000**	**0.0333**
	LOF	0.9900	0.9859	1.0000	0.9929	1.0000	0.0000	0.0333
	iFOREST	0.9900	0.9859	1.0000	0.9929	1.0000	0.0000	0.0333
AutoTAB	OCSVM	0.6900	0.9535	0.5857	0.7257	0.8100	0.4143	0.0667
	LOF	0.5900	0.8085	0.5429	0.6496	0.6900	0.4571	0.3000
	iFOREST	0.7900	0.9623	0.7286	0.8293	0.9300	0.2714	0.0667

4.9.1.2 Test 2: CERT r5.2 Dataset

The Transformer model again demonstrated superior anomaly detection, with OCSVM leading in performance metrics, achieving an accuracy of 93.08%, precision of 90%, and an F1 score of 94.74%. However, that comes at the cost of a higher FPR of 18.33%. Conversely, the AutoTAB model exhibits lower efficiency across all metrics, with iFOREST showing relative improvement, achieving an accuracy of 69.18% and an F1 score of 69.94%. The disparity in FPR and FNR between the two models highlights the Transformer's enhanced capability, particularly with the OCSVM algorithm, in navigating the balance between identifying true positives and minimizing false alarms. Table 4.5 depicts the performance difference between the Transformer and AutoTAB using Test-2 based on the CERT r5.2 dataset.

Table 4.5: *Test-2: Transformer vs. AutoTAB Performance Metrics*

Model	Detection Method	A	P	R	F1	AUC	FNR	FPR
Trans	OCSVM	**0.9308**	**0.9000**	**1.0000**	**0.9474**	**0.9000**	**0.0000**	**0.1833**
	LOF	0.9245	0.8919	1.0000	0.9429	0.8800	0.0000	0.2000
	iFOREST	0.9245	0.8919	1.0000	0.9429	0.9000	0.0000	0.2000
AutoTAB	OCSVM	0.6101	0.9111	0.4141	0.5694	0.7600	0.5859	0.0667
	LOF	0.5597	0.8085	0.3838	0.5205	0.6200	0.6162	0.1500
	iFOREST	0.6918	0.8906	0.5758	0.6994	0.8400	0.4422	0.1167

4.9.1.3 Test 3: CERT r6.2 Dataset

Test-3, which is based on CERT r6.2, has a severe imbalance between its benign and malicious users, 120 and Five, respectively. Despite that, the Transformer model again outperforms the AutoTAB. The OCSVM and LOF methods within the Transformer framework both exhibit a high accuracy

of 0.9440% and a perfect recall of 100%. However, their precision and F1 scores are notably lower at 41.67% and 58.82%, respectively. In contrast, the AutoTAB model's performance is less optimal, with OCSVM achieving an accuracy of 92% but lower scores across all other metrics. Table 4.6 depicts the performance difference between Transformer and AutoTAB using Test-3 based on the CERT r6.2 dataset.

Table 4.6: *Test-3: Transformer vs. AutoTAB Performance Metrics*

Model	Detection Method	A	P	R	F1	AUC	FNR	FPR
Trans	OCSVM	**0.9440**	**0.4167**	**1.0000**	**0.5882**	**0.9800**	**0.0000**	**0.0583**
	LOF	0.9440	0.4167	1.0000	0.5882	0.9700	0.0000	0.0583
	iFOREST	0.9200	0.3333	1.0000	0.5000	0.9700	0.0000	0.0833
AutoTAB	OCSVM	0.9200	0.2727	0.6000	0.3750	0.8400	0.4000	0.0667
	LOF	0.8720	0.0769	0.2000	0.1111	0.7600	0.8000	0.1000
	iFOREST	0.8320	0.1667	0.8000	0.2759	0.9100	0.2000	0.1667

4.9.1.4 Test 4: Aggregated Dataset

As illustrated in Table 4.2, Tests-1 through Test-3 are designed to evaluate the three different CERT datasets individually. However, this introduced some limitations; for instance, in Test-3, there were only five malicious users compared to 120 benign ones, which skewed the results from some metrics. To address this problem and ensure a more balanced and representative evaluation, we designed Test-4, which amalgamates users across all CERT dataset versions, culminating in a robust test set consisting of 210 benign and 174 malicious users. The results from Test 4, detailed in Table 4.7, affirm the benefits of this inclusive testing strategy.

Unsurprisingly, after rerunning Test-4, the Transformer again outper-

formed the AutoTAB encoder across all detection methods and metrics. It achieves an accuracy of 96.61%, a recall of 100%, and an FPR of 0.0571. In contrast, AutoTAB's best performance came from the iFOREST algorithm, registering an accuracy of only 75.52%, an F1 score of 73.45%, and an FPR of 16.67%. This inclusive test highlights a notable disparity in performance between the transformer and autoTAB, which unequivocally underscores the Transformer model's superior precision and reliability enabled by our UBS structure. Table 4.7, depicts this inclusive testing result between the Transformer and AutoTAB using our consolidated Test-4, which is based on CERT r4.2, r5.2

Table 4.7: *Test-4: Transformer vs. AutoTAB Performance Metrics*

Model	Detection Method	A	P	R	F1	AUC	FNR	FPR
Trans	OCSVM	**0.9505**	**0.9016**	**1.0000**	**0.9482**	**0.9600**	**0.0000**	**0.0905**
	LOF	0.9141	0.9325	0.8736	0.9021	0.9500	0.1264	0.0524
	iFOREST	0.9661	0.9351	0.9943	0.9638	0.9500	0.0057	0.0571
AutoTAB	OCSVM	0.7318	0.8586	0.4885	0.6227	0.7900	0.5115	0.0667
	LOF	0.6953	0.7822	0.4540	0.5745	0.6700	0.5460	0.1048
	iFOREST	0.7552	0.7222	0.7471	0.7345	0.8600	0.2529	0.2381

Figure 4.11 graphically illustrates the difference in the recall, FNR, and FPR between the Transformer and Autoenoder-TAB using the aggregated Test-4. The ideal recall rate is "1", indicating perfect detection of positive instances. Similarly, the False Negative Rate (FNR) and False Positive Rate (FPR) should ideally be close to zero, reflecting minimal classification errors. The Transformer by far exceeds the Autoencoder-TAB in all three metrics.

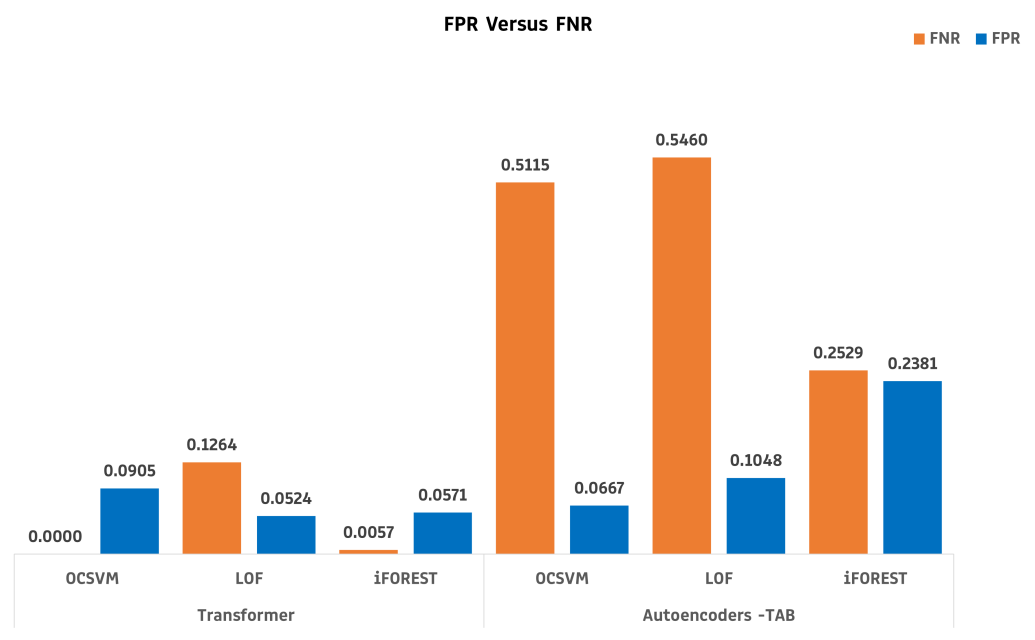


Figure 4.11: *Test-4: Transformers vs. AutoTAB (FNR/FPR) Graph*

4.9.2 Transformer Versus AutoUBS

In this comparison, we focus exclusively on Test-4 because it offers a more comprehensive basis to illustrate the effectiveness of our UBS structure. Specifically, it demonstrates how the previously discussed Autoencoder achieves significantly better results when applied to user-based sequencing data, as opposed to tabular data.

4.9.2.1 Test 4: Aggregated Dataset

As anticipated, AutoUBS has delivered exceptional results across all detection algorithms. Notably, the iForest algorithm has achieved an accuracy of 96.88%, a precision of 98.21%, an F1 score of 96.49%, and an FNR of 5.17%. These metrics are on par with those of the Transformer model, which only slightly surpasses AutoUBS in terms of recall and FNR. By comparing the Transformer and Autoencoder using UBS, we can clearly

see the pivotal role that our UBS structure plays in enhancing the detection capabilities and efficiency of the Autoencoder. Table 4.8 depicts this exceptional performance when running the Autoencoder on UBS structure.

Table 4.8: *Transformer vs. AutoUBS Performance Metrics (All CERTs)*

Model	Detection Method	A	P	R	F1	AUC	FNR	FPR
	OCSVM	**0.9505**	**0.9016**	**1.0000**	**0.9482**	**0.9600**	**0.0000**	**0.0905**
Trans	LOF	0.9141	0.9325	0.8736	0.9021	0.9500	0.1264	0.0524
	iFOREST	0.9661	0.9351	0.9943	0.9638	0.9500	0.0057	0.0571
	OCSVM	0.9427	0.9143	0.9770	0.9392	0.9900	0.0230	0.0857
AutoUBS	LOF	0.9609	0.9818	0.9310	0.9558	1.0000	0.0690	0.0143
	iFOREST	0.9688	0.9821	0.9483	0.9649	1.0000	0.0517	0.0143

Figure 4.12 graphically illustrates the difference in recall, FNR, and FPR between the Transformer and Autoenoder-UBS using the aggregated Test-4.

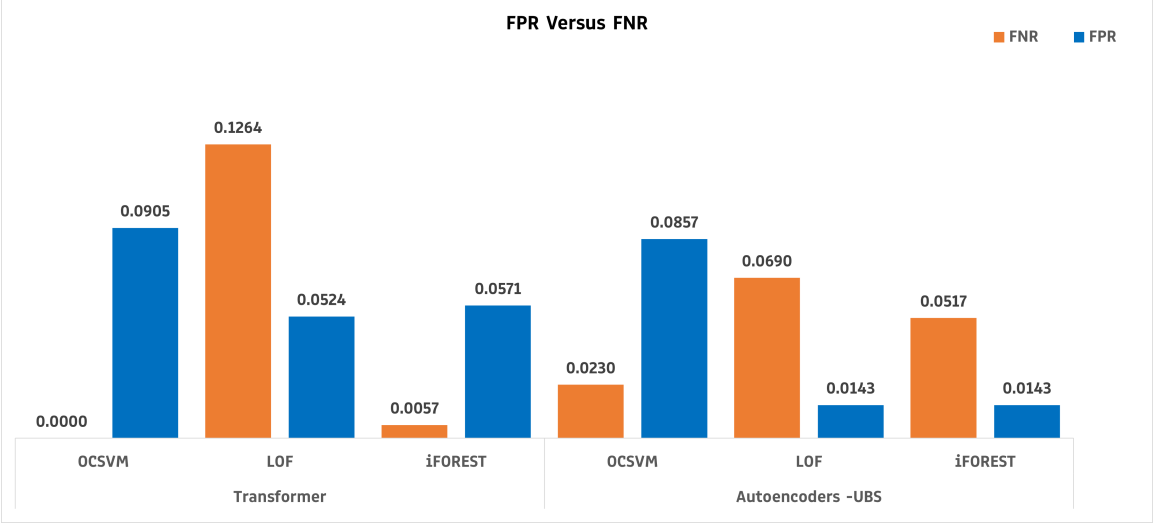


Figure 4.12: *Test-4: Transformers vs. AutoUBS (FNR/FPR) Graph*

4.10 AutoUBS Versus AutoTAB Results

4.10.1 Test 4: Combined Dataset

In order to evaluate Hypothesis 2, which proposes that user-based sequencing enhances the model's performance, we decided to compare the AutoTAB and AutoUBS using "Test 4". As anticipated, the AutoUBS outperformed the AutoTAB, with iForest in the AutoUBS achieving an accuracy of 96.88%, precision of 98.21%, an F1 score of 96.49%, and an FNR of 5.17% compared to the AutoTAB achieving an accuracy of 75.52%, a precision of 72.22%, an F1 score of 73.45%, and an FNR of 25.29%. This comparison underscores our UBS structure's important role in improving model performance and efficiency. The result of this test is summarized in Table 4.9.

Table 4.9: *Summary of Results from Test 4*

Model	Detection Method	A	P	R	F1	AUC	FPR
Auto-TAB	OCSVM	0.7318	0.8586	0.4885	0.6227	0.7900	0.0667
	LOF	0.6953	0.7822	0.4540	0.5745	0.6700	0.1048
	iFOREST	0.7552	0.7222	0.7471	0.7345	0.8600	0.2381
Auto-UBS	OCSVM	0.9427	0.9143	0.9770	0.9392	0.9900	0.0857
	LOF	0.9609	0.9818	0.9310	0.9558	1.0000	0.0143
	iFOREST	0.9688	0.9821	0.9483	0.9649	1.0000	0.0143

Figure 4.13 graphically illustrates the difference in recall, FNR, and FPR between the AutoUBS and AutoTAB using the aggregated Test-4.

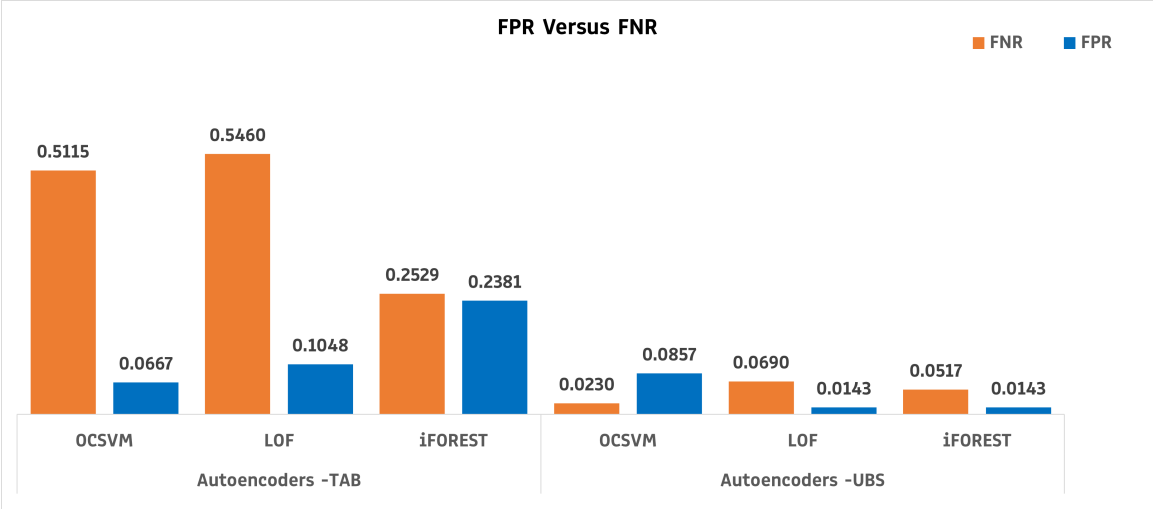


Figure 4.13: *Test-4: AutoUBS vs. AutoTAB (FNR/FPR) Graph*

4.11 Consolidated Test Results

This section consolidates all the previously discussed test results in one place for ease of reference. Tables 4.10 and 4.11 summarize the results.

Table 4.10: *Summary of Results from Test-1, 2, and 3*

Model	Detection Method	A	P	R	F1	AUC	FNR
			Test-1 (CERT r4.2)				
Transformer	OCSVM	**0.9900**	0.9859	**1.0000**	0.9929	1.0000	**0.0000**
	LOF	0.9900	0.9859	1.0000	0.9929	1.0000	0.0000
	iFOREST	0.9900	0.9859	1.0000	0.9929	1.0000	0.0000
Auto-TAB	OCSVM	0.6900	0.5857	0.5857	0.7257	0.8100	0.4143
	LOF	0.5900	0.8085	0.5429	0.6496	0.6900	0.4571
	iFOREST	0.7900	0.9623	0.7286	0.8293	0.9300	0.2714
Auto-UBS	OCSVM	0.9700	1.0000	0.9571	0.9781	0.9981	0.0429
	LOF	0.9100	1.0000	0.8714	0.9313	1.0000	0.1286
	iFOREST	0.9300	1.0000	0.9000	0.9474	1.0000	0.1000
			Test-2 (CERT r5.2)				
Transformer	OCSVM	0.9308	0.9000	**1.0000**	0.9474	0.9000	**0.0000**
	LOF	0.9245	0.8919	1.0000	0.9429	0.8800	0.0000
	iFOREST	0.9245	0.8919	1.0000	0.9429	0.9000	0.0000
Auto-TAB	OCSVM	0.6101	0.9111	0.4141	0.5694	0.7600	0.5859
	LOF	0.5597	0.8085	0.3838	0.5205	0.6200	0.6162
	iFOREST	0.6918	0.8906	0.5758	0.6994	0.8400	0.4422
Auto-UBS	OCSVM	0.9434	0.9245	0.9899	0.9561	0.9806	0.0101
	LOF	0.9245	0.9143	0.9697	0.9412	0.9557	0.0303
	iFOREST	0.9308	0.9151	0.9798	0.9463	0.9286	0.0202
			Test-3 (CERT r6.2)				
Transformer	OCSVM	**0.9440**	0.4167	**1.0000**	0.5882	0.9800	**0.0000**
	LOF	0.9440	0.4167	1.0000	0.5882	0.9700	0.0000
	iFOREST	0.9200	0.3333	1.0000	0.5000	0.9700	0.0000
Auto-TAB	OCSVM	0.9200	0.2727	0.6000	0.3750	0.8400	0.4000
	LOF	0.8720	0.0769	0.2000	0.1111	0.7600	0.8000
	iFOREST	0.8320	0.1667	0.8000	0.2759	0.9100	0.2000
Auto-UBS	OCSVM	0.9200	0.3333	1.0000	0.5000	1.0000	0.0000
	LOF	0.9360	0.3846	1.0000	0.5556	0.9967	0.0000
	iFOREST	0.9360	0.3846	1.0000	0.5556	0.9917	0.0000

Table 4.11: *Summary of All Results from Test-4*

Model	Method	A	P	R	F1	AUC	FPR
Test-4 (All CERT datasets combined)							
Transformer	OCSVM	0.9505	0.9016	**1.0000**	0.9482	0.9600	**0.0905**
	LOF	0.9141	0.9325	0.8736	0.9021	0.9500	0.0524
	iFOREST	**0.9661**	0.9351	**0.9943**	0.9638	0.9500	**0.0571**
Auto-TAB	OCSVM	0.7318	0.8586	0.4885	0.6227	0.7900	0.0667
	LOF	0.6953	0.7822	0.4540	0.5745	0.6700	0.1048
	iFOREST	0.7552	0.7222	0.7471	0.7345	0.8600	0.2381
Auto-UBS	OCSVM	0.9427	0.9143	0.9770	0.9392	0.9900	0.0857
	LOF	0.9609	0.9818	0.9310	0.9558	1.0000	0.0143
	iFOREST	0.9688	0.9821	0.9483	0.9649	1.0000	0.0143

4.12 Analysis & Discussion

One of the primary goals we set in this research is to compare the effectiveness of the Transformer using our novel UBS to our baseline model—Autoencoder—which uses unstructured tabular data. To achieve this goal, we built four test sets (1 through 4) as documented in Table 4.2 and developed three models (Transformer using UBS, Autoencoder using Tabular, and Autoencoder using UBS)

We also built a fourth baseline model using OCSVM, LOF, and IFORST. However, we quickly realized that those models are ineffective as the data structure (UBS or tabulated) is not designed to accommodate such models. Therefore, we elect not to document their performance in this study. Furthermore, we also thought of running the Transformer on tabular data. However, again, we realized that the Transformer architecture is not de-

signed to work well in non-sequential data. Hence, we abandon that idea early in our research.

Every test we have designed is based on a stratified splitting approach to ensure complete data separation and that the data in each set are unique; no overlap is present in any of the sets. Among all the test sets we built, Test-4 is the most comprehensive, combining user data from all CERT dataset versions.

As mentioned in the previous sections, we used several metrics, including accuracy, precision, F1, and recall, to measure the performance of the three anomaly detection algorithms employed in this study. However, given the severity of insider threats and the unique anomaly approach we designed, we decided that reducing the false negatives and maximizing recall should be prioritized over other metrics.

The results obtained by stacking the Transformer against both versions of the autoencoders are summarized in Tables 4.10 and 4.11. As previously demonstrated, the Transformer exhibited exceptional performance, outperforming all baseline models and comparable models reviewed in Chapter 2. Notably, our model achieved a recall of 99.43%, surpassing the Distilled-Trans introduced by Wang and El Saddik (2023), which achieved a recall of 84.62%. Additionally, our Transformer model outperformed the TRANLOG introduced by Guo et al. (2021), which utilizes a similar anomaly approach and achieved only a 0.996% recall.

As demonstrated in Table 4.11, using Test-4 as the defacto test set, Transformer (OCSVM) stands out for its exceptional performance in recall and FNR, achieving a perfect score of 100% and 0.0%, respectively. These metrics are particularly relevant in insider threat scenarios where missing a true positive can have serious consequences. Despite the iFOREST's high

accuracy and F1-score, the flawless recall and zero FNR of OCSVM make it the most effective algorithm.

- **Transformer (iFOREST) vs. Auto-TAB:**

  The Transformer OCSVM exhibits exceptional performance when compared to Auto-TAB. The Transformer model has delivered a remarkable 33.85% improvement in recall over the best-performing Auto-TAB model (iFOREST). This notable enhancement underscores the Transformer OCSVM's capability to identify all positive instances without any misses, as evidenced by its perfect recall rate and 0.0% FNR. Table 4.12 provides a clear breakdown of the percentage improvement of the Transformer over the Auto-TAB.

Table 4.12: *Improvement of Transformer over AutoTAB*

Metrics	% Point Change	Analysis
Accuracy	+27.92%	Substantial increase in overall model performance.
Precision	+29.48%	Notable improvement in true positive identifications.
Recall	+33.10%	Significant enhancement in identifying all actual positives.
F1 Score	+31.22%	Considerable betterment in the balance between precision and recall.
AUROC	+10.47%	Improved capability in distinguishing between the classes.
FPR	+ 76.01%	Reduction in the rate at which true positives are mistakenly overlooked.

- **Transformer (iFOREST) vs. Auto-UBS:**

  Despite the impressive improvements and high scores shown by the Auto-UBS model, particularly in AUROC and Precision, the flawless Recall and FNR demonstrated by the Transformer OCSVM are critical for applications where missing an anomaly could have dire consequences.

The Transformer model with iFOREST boasts a 4.85% enhancement in recall over the top-performing Auto-UBS model (iFOREST). This improvement underscores the effectiveness of the Transformer iFOREST in achieving a perfect recall rate, surpassing the already impressive recall performance of Auto-UBS. Table 4.13 provides an overview of the percentage improvement of the Transformer compared to Auto-UBS.

Table 4.13: *Improvement of Transformer over AutoUBS*

Metrics	% Point Change	Analysis
Accuracy	-0.28%	A slight enhancement in the overall model performance by the Transformer over Auto-UBS.
Precision	-4.79%	A minor reduction in the proportion of true positive identifications over all positive predictions by the Transformer compared to Auto-UBS.
Recall	+4.85%	Superior ability of the Transformer to identify all actual positives.
F1 Score	-0.11%	A slightly better balance between precision and recall by the Transformer.
AUROC	-5.00%	A small decline in the Transformer's ability to distinguish between the classes compared to Auto-UBS.
FPR	4.27	(absolute) increase in the rate at which true positives are mistakenly overlooked by the Transformer compared to Auto-UBS.

The impressive performance of both the Transformer and Auto-UBS underscores the critical role our user-based sequencing plays in enabling both models to achieve better results by transforming the data so that each model can understand the data and extract meaningful patterns more effectively. The Transformer, with its ability to handle sequential data and capture long-range dependencies, and the Autoencoder, known for its feature extraction and reconstruction capabilities, both benefit immensely from the user-based sequencing approach.

- **AutoUBS vs. AutoTAB:**

  The Auto-UBS (iFOREST) shows notable improvements over the Auto-TAB (iFOREST) model across all metrics, affirming the benefit of our user-based sequencing in improving the Autoencoder detection capabilities. Table 4.14 shows the percentage of improvement of the Auto-UBS over the Auto-TAB.

Table 4.14: *Improvement of AutoUBS over AutoTAB*

Metrics	% Point Change	Analysis
Accuracy	+28.28%	A significant enhancement in overall model performance.
Precision	+35.97%	A better proportion of true positive identifications over all positive predictions.
Recall	+26.94%	A superior ability to identify all actual positives.
F1 Score	+31.31%	A better balance between precision and recall.
AUROC	+16.28%	An improved capability in distinguishing between the classes.
FPR	+ 93.99%	A substantial reduction in the rate at which true positives are mistakenly overlooked.

The above improvement underscores the substantial contribution our user-based sequencing made to enable the Auto-UBS to enhance its detection capabilities to identify anomalies compared to the Auto-TAB using Tabular data format.

4.12.1 Our Model versus Others

In Table 4.15, we have compared the performance of our Transformer with fifteen other baselines found in the literature for Insider Threat Detection. The top two results are highlighted, with the best result being boldfaced and the second-best result being underlined. An upward arrow indicates

when the metrics have a higher value, it is better, while a downward arrow indicates that a smaller number is better (Cai et al., 2024).

Table 4.15: *Comparison to Other Baseline Models*

Model	DR (Recall) ↑	FPR (%)↓	FNR ↓
Original Transformer [+1] (Wang and El Saddik, 2023)	80.77	X	0.1923
DistilledTrans [+2] (Wang and El Saddik, 2023)	84.62	X	0.1538
BERT+FL [+3] (Wang and El Saddik, 2023)	95.38	X	0.0462
Roberta+FL [+4] (Wang and El Saddik, 2023)	94.62	X	0.0538
CAE (Wei et al., 2021)	92.50	0.0510	0.0750
Stack CNN (Anju and Krishnamurthy, 2024)	95.00	X	0.0500
LSTM-AD (Villarreal-Vasquez et al., 2023)	97.29	0.0380	0.0271
LAN (Cai et al., 2024)	94.78	0.0120	0.0522
ITDBERT: Bi-LSTM (Huang et al., 2021)	91.87	X	0.0813
Deep-Learning (Nasir et al., 2021)	92.00	0.0900	0.0800
Stacked BiLSTM & FNN (Song et al., 2024)	90.70	X	0.0930
AD-DNN (Al-Mhiqani et al., 2021)	95.00	0.0400	0.0500
Our Transformer(iFOREST)	**99.43**	**0.0571**	**0.0057**

Based on the data shown in Table 4.15, our Transformer model has demonstrated superior performance compared to baseline models across important metrics such as Recall and FPR. Our model achieved a DR (Recall) of 99.43% and 100% using iForest and OCSVM, respectively, outperforming the transformer baseline models by significant margins (23.81% [+1], 18.17% [+2], 4.84% [+3], and 5.6% [+4]). Unlike the baseline models, which relied solely on a single version of the CERT dataset, our results were derived from a combination of datasets, including CERT r4.2, r5.2, and r6.2. Additionally, we included FPR metrics in our evaluation, achieving rates between 0.0905% and 0.0571% using OCSVM and iFOREST detection algorithms, respectively. We believe that the omission of FPR in the baseline assessments is a significant oversight, given its critical relevance in the context of Insider Threat detection.

4.12.2 Model Training & Inference Timing

When we evaluate the Transformer and Autoencoder timing performance, as demonstrated in Table 4.16, each model exhibits distinct timing differences for training and inference based on the number of epochs and test size. The Transformer model requires 12 minutes and 31 seconds for training and approximately 2.22 seconds for inference, while the Autoencoder-UBS takes 16.42 minutes to train and 3.30 minutes for inference. The Autoencoder-TAB on the other hand, takes 35.68 minutes to train and 1.98 seconds for inference. The longer inference time of the Transformer and Autoencoder-UBS are due to thier processing of user-based sequencing data, where it calculates the reconstruction error for each of the 35 features in three-dimensional data. In contrast, the Autoencoder-TAB processes simpler, one-dimensional tabular data and computes a single reconstruction error, thus requiring less time during inference. Despite its longer inference time, the Transformer and Autoencoder-UBS significant shorter training duration suggests that their approach of using user-based sequencing data accelerates the learning process, effectively balancing the trade-offs between computational complexity and operational speed. However, as we demonstrated in Table 4.12 the Autoencoder-TAB's results are notably less accurate, lagging by 33.85%, 25.86%, and 24.84% in recall, accuracy, and precision, respectively, compared to the Transformer as well as the autoencoder-UBS. These results highlight the importance of considering both training efficiency and inference performance, as well as the model's accuracy and overall effectiveness.

Table 4.16: *Models Training and Inference Times*

Model	Training Time	Inference Time
Transformer	**12:31** minutes	2.22 seconds
Autoencoder-TAB	35.68 minutes	**1.98** seconds
Autoencoder-UBS	16.42 minutes	3.30 seconds

4.13 Evaluating the Validity of Hypotheses

In order to determine whether the differences in performance between our models were statistically significant, we used the McNemar test (Pembury Smith & Ruxton, 2020). This is a non-parametric method that is used to compare two classification models on the same dataset, focusing on instances where their predictions disagree (Raschka, 2018). We used both the corrected and exact p-values when applying the McNemar test. The corrected p-value, which is often used to adjust for continuity in sample sizes, provides a more conservative estimate of significance and reduces the likelihood of Type I errors. On the other hand, the exact p-value, derived from the exact binomial distribution of the test's statistics, offers precise significance levels without requiring large sample approximations.

In Figure 4.14, we investigated the null hypothesis asserting that Transformer encoders and autoencoders are equally effective in identifying malicious insider threats, which aligns with our initial hypothesis in chapter one that "Transformer encoders can better capture long-range dependencies than Autoencoders and accurately identify malicious insider threats." Using the McNemar test, as depicted in Figure 4.14, we compared the transformer accuracy of 0.9505 and Autoencoder-tabular accuracy of 0.7318 using Test-

4, as documented in Table 4.11. Based on the result obtained from our top-performing OCSVM detection algorithm, we are able to reject the Null Hypothesis using both the corrected and exact p-values (which are less than our predetermined significance level, α, set at 0.05). Figure 4.14 displays the McNemar contingency table, the calculated Chi-square, and the exact and corrected p-values used for testing the hypothesis. Figure 4.14 solely presents the test result from the OCSVM. Nevertheless, we obtained the same result from our LOF and IFOREST detection algorithms (The results from these tests are documented in Appendix B Figure B.1, B.2, B.3).

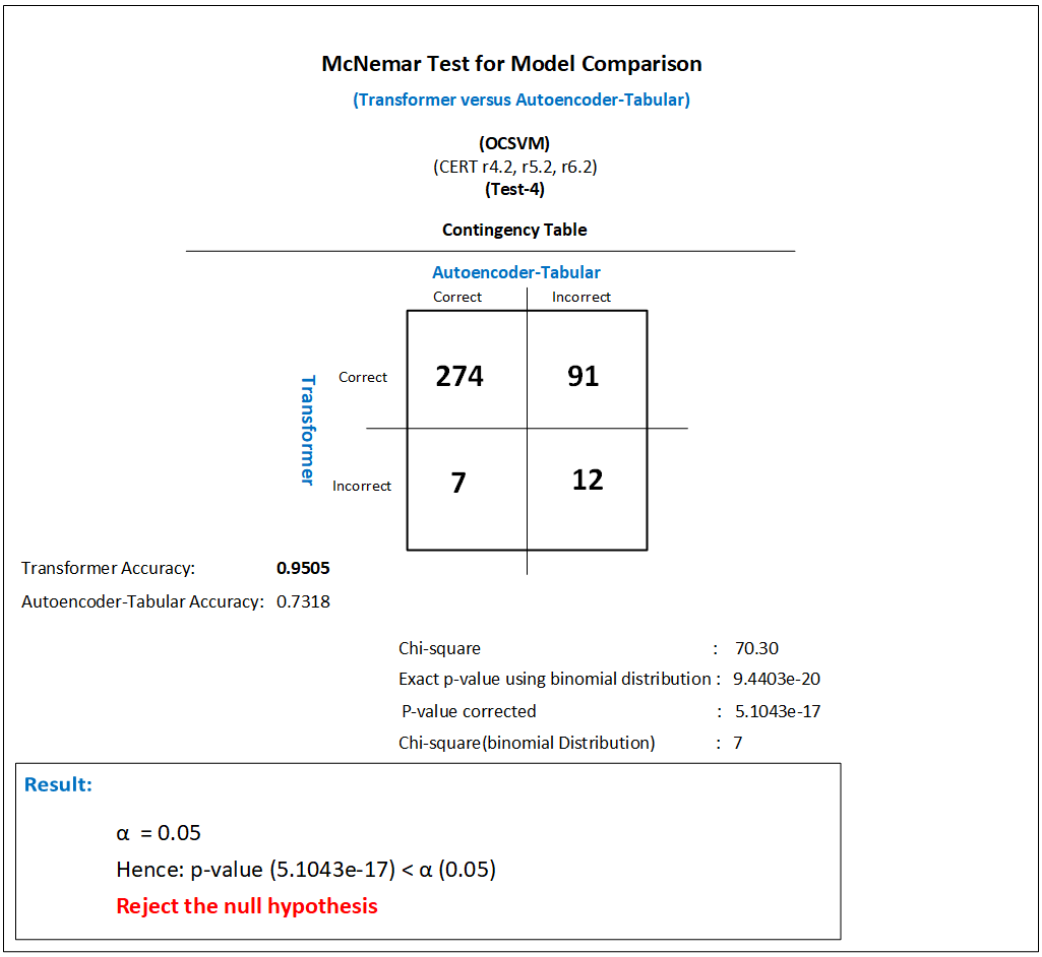


Figure 4.14: *McNemar: Transformer vs. Autoencoder-Tabular*

In Figure 4.15 and 4.16, we examined our second and third hypothe-

ses from Chapter One, which suggest that using user-based sequencing improves the implementation of a Transformer Encoder on tabular data, and employing feature optimization, such as UBS, enhances the speed and detection rate across different CERT dataset versions (which we already documented in Table 4.16). However, in 4.15, we compared the Autoencoder-UBS accuracy of 0.9427 and the Autoencoder-tabular accuracy of 0.7318 using Test-4, as outlined in Table 4.11. Based on the results obtained from our top-performing OCSVM detection algorithm, we rejected the Null Hypothesis using both the corrected and exact p-values at a pre-determined significance level of 0.05. Figure 4.15 illustrates the McNemar contingency table, the computed Chi-square, and the exact and corrected p-values used for hypothesis testing. Similar results from LOF and IFOREST are documented in Appendix B, Figure B.1, B.2, and B.3.

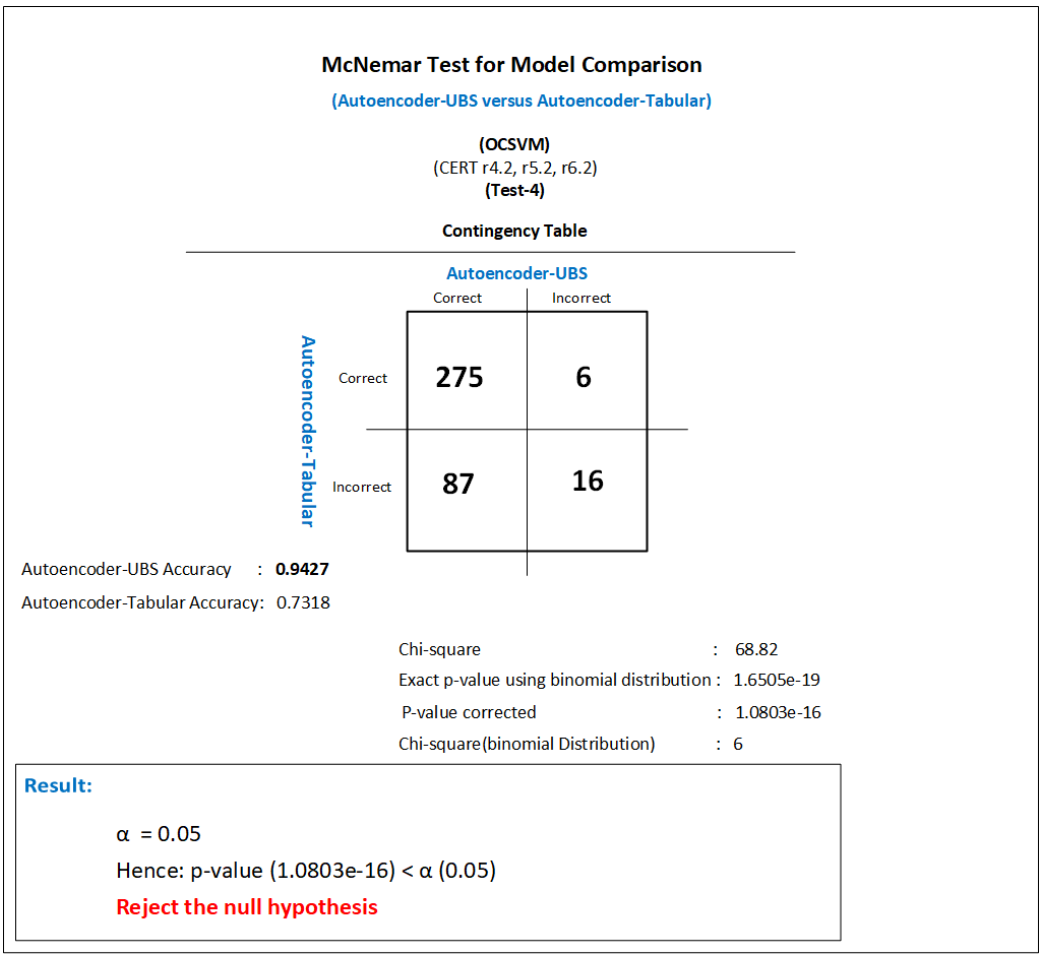


Figure 4.15: *McNemar: Transformer vs. Autoencoder-Tabular*

In the analysis shown in Figure 4.16, we compared the Transformer and the Autoencoder with the Autoencoder utilizing our innovative User-Based Sequencing method. In contrast to the results obtained when running the Autoencoder on tabular data, employing the Autoencoder with UBS led to improved model performance when compared to the Transformer. However, while the Transformer showcased better performance than the Autoencoder-UBS, the difference was not substantial. As a result, our McNemar test, which compared the Transformer's accuracy of 0.9505 with the Autoencoder-UBS's accuracy of 0.9427, did not provide enough evidence to reject the hypothesis that both models perform similarly. Similar findings

from LOF and IFOREST can be found in Appendix B, specifically in Figures B.1, B.2, and B.3.

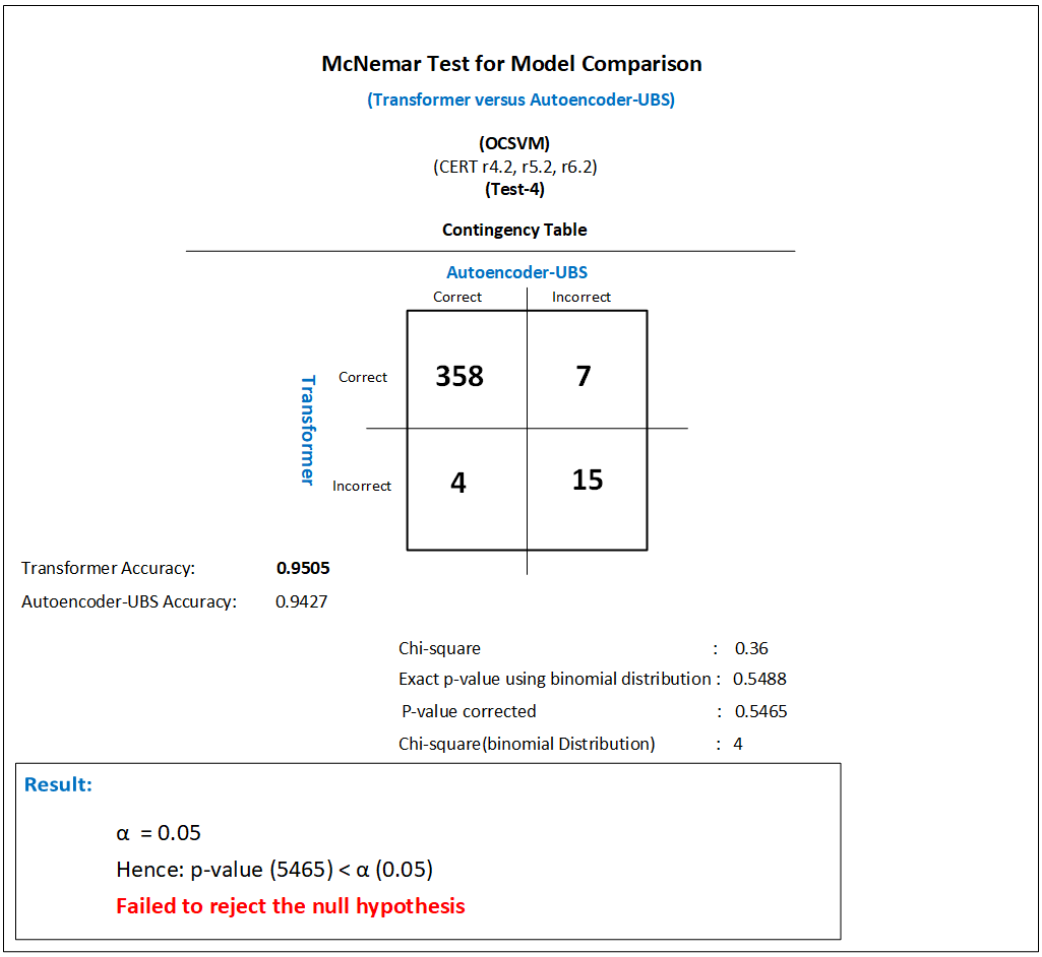


Figure 4.16: *McNemar: Transformer vs. Autoencoder-Tabular*

4.13.1 Interpreting Results

As mentioned in Chapter 3, we employed the Transformer model to generate reconstruction errors, which were then fed into the detection algorithms to determine whether a particular record was anomalous. While deep learning models like the Transformer are often considered "black boxes," and challenging to explain, our architecture made using explainable AI to interpret the Transformer decision even more challenging. Nonetheless,

we could use Shapley Additive explanations (SHAP) to explain the decision made by the detection algorithms based on the reconstruction errors fed to them by the Transformer model.

To set up the SHAP test case, we first trained the anomaly detection algorithms—IFOREST, LOF, and OCSVM—using Test-4. Following this, we initiated an Explainer object for two randomly selected rows: Row 3 represents a benign user and Row 270 represents a malicious user. Figure 4.17 depicts the model prediction, which matches the ground truth for row 3 and row 270.

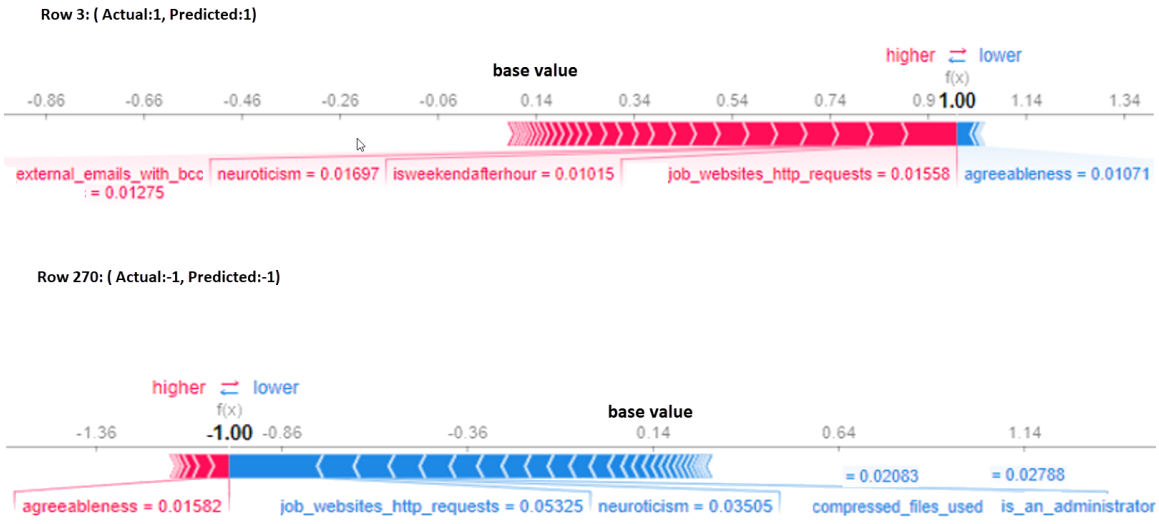


Figure 4.17: SHAP Explainer Object Graph

Features highlighted in RED illustrate their contribution towards increasing the prediction's likelihood, whereas features in BLUE indicate their role in reducing the prediction's value. We run a waterfall plot to facilitate a more straightforward interpretation of the above graph. The outcome is presented in Figure 4.18.

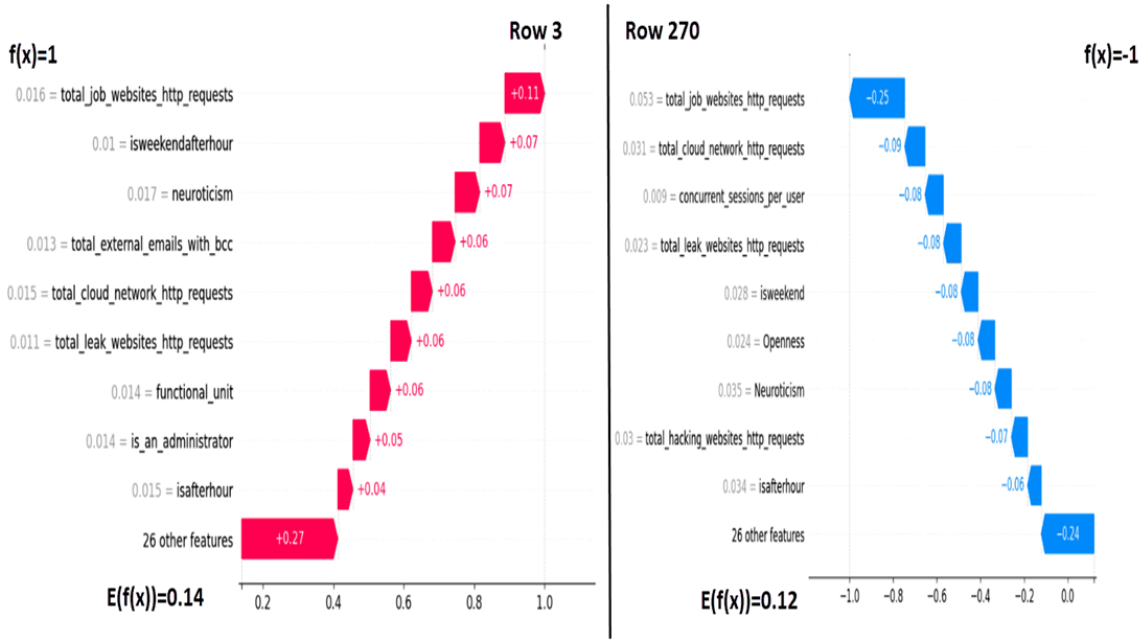


Figure 4.18: SHAP Waterfall Graph

Finally, leveraging SHAP analysis, we investigate the global impact of features on our model's predictions. Figure 4.19 showcases the relative importance of these features, as elucidated by SHAP values

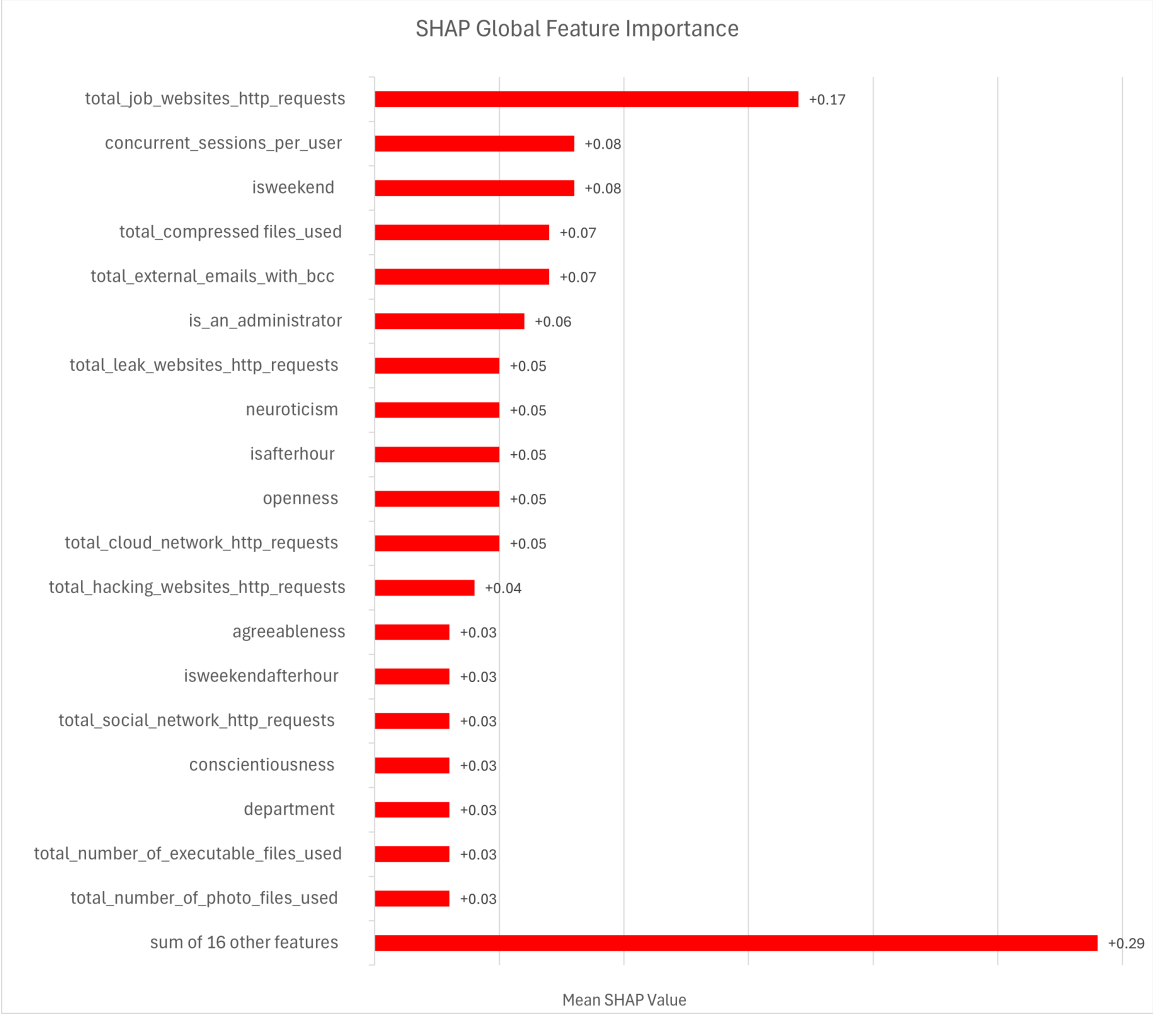


Figure 4.19: Features Importance

4.13.2 Conclusion

After comparing the performance metrics of the Transformer and the Autoencoder, it is evident that the Transformer OCSVM has achieved a significant milestone by attaining a 100% recall and 0% FNR. This is especially important in environments where detecting every potential threat is critical. However, it is worth noting that our User-Based Sequence has benefited not only the Transformer but also the Autoencoder, enabling it to achieve better results when compared to the Autoencoder using tabular data.

The results from this Chapter address the research questions and hypotheses outlined in Chapter 1 and repeated below:

RQ1: How do Transformer encoders compare to Autoencoders in terms of accuracy, precision, and recall when detecting malicious insider threats?

RQ2: How does user-based sequencing improve the accuracy of insider threat detection?

RQ3: How can a model based on transformer encoders be optimized to enhance both speed and detection rate across various versions of the CERT insider threat datasets?

H1: Transformer encoders can capture long-range dependencies better than Autoencoders and accurately detect malicious insider threats.

H2: User-based sequencing enables the implementation of a Transformer Encoder on tabular data to improve the detection of insider threats.

H3: Using feature optimization will improve the speed and detection rate across various CERT dataset versions.

Chapter 5: Discussion and Conclusions

5.1 Conclusion

Insider threats are a challenging problem for organizations of all kinds. The problem is further exacerbated because of the secretive nature of these threats, limited reporting mechanisms, and strict privacy laws, which led to a lack of sufficient data for research. Despite these challenges, over the two last decades, researchers have made some progress in developing Machine-Learning (ML) solutions to identify, prevent, and reduce insider risks. However, our literature review in Chapter 2 identified a significant gap in using Transformer Encoders in insider threat detection despite their proven success in other cyber-security fields. This research addresses this gap by building a predictive model using the Transformer architecture, novel User-Based Sequencing (UBS), and ML anomaly detection algorithms to improve insider threat detection capabilities.

In Chapter 3, we elaborated on the methodology we employed to implement our proposed experiment, which is based on the principles of anomaly detection. Our methodology follows a structured approach for collecting and understanding the data, preparing and selecting the relevant features, designing and creating UBS structure, developing and building the model architectures, and training, testing, and evaluating the model performance.

In Chapter 4, we showcased the impressive performance of our stratified testing approach using test sets based on individual CERT dataset versions, such as r4.2, r5.2, and r6.2, as well as a mixture of data from all CERT dataset versions. Our Transformer model achieved state-of-the-art performance, with an accuracy rate of 96.61%, a recall of 99.43%, an F1-score

of 96.38%, an AUC-ROC value of 95.00%, and a False Negative Rate (FNR) and False Positive Rate (FPR) of 0.0057 and 0.0571, respectively.

5.2 Contribution to the Body of Knowledge

Firstly, we developed an innovative technique called User-Based Sequencing to organize session vectors into user-specific sequences by transforming unstructured tabular data into a structured format that is optimally suited for the Transformer to capture temporal and sequential patterns. Secondly, we integrated User-Based Sequencing with the Autoencoder to improve its anomaly detection capabilities. Thirdly, we innovatively incorporated machine learning algorithms such as One Class Support Vector Machine (OCSVM),Local Outlier Factor (LOF), and Isolation Forest (IFOREST) in our methodology pipeline to enhance the model anomaly detection capabilities and enable additional performance metrics generation such as Receiver Operating Characteristic (ROC) and anomaly scores.

5.3 Recommendations for Future Research

In order to enable a better grasp of the Transformer's decision-making mechanisms, we highly recommend using some "Explainable AI" to explore the interpretability of the Transformer model. In this research, we only used "Explainable AI" to interpret the decision from the detection algorithms rather than the Transformer model itself. Additionally, it is imperative to validate our research outcomes using real-world data, as our findings were derived from the synthetic CERT datasets. While these datasets are useful for development and testing, they may not fully capture the complexity of user behavior in the context of insider threat detection.

References

Agesen, O. (1995). The cartesian product algorithm: Simple and precise type inference of parametric polymorphism. *ECOOP'95—Object-Oriented Programming, 9th European Conference, Åarhus, Denmark, August 7–11, 1995 9*, 2–26.

Agrafiotis, I., Nurse, J. R., Buckley, O., Legg, P., Creese, S., & Goldsmith, M. (2015). Identifying attack patterns for insider threat detection. *Computer Fraud and Security*, *2015*(7), 9–17. http://dx.doi.org/10.1016/S1361-3723(15)30066-X

Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., & Ahmad, F. (2021). Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, *32*(1), e4150.

Alammar, J. (2018). The illustrated transformer [Accessed: 2024-06-14].

Al-Mhiqani, Ahmad, R., Zainal Abidin, Z., Mohamed, W., Hassan, A., Abdulkareem, K., Ali, N., & Yunos, Z. (2020). A review of insider threat detection: Classification, machine learning techniques, datasets, open challenges, and recommendations. *Applied Sciences*, *10*, 1–41. https://doi.org/10.3390/app10155208

Al-Mhiqani, Ahmed, R., Zainal Abidin, Z., & Isnin, S. (2021). An integrated imbalanced learning and deep neural network model for insider threat detection. *International Journal of Advanced Computer Science and Applications*, *12*(1), 573–577. http://dx.doi.org/10.14569/IJACSA.2021.0120166

Al-Shehari, T., Al-Razgan, M., Alfakih, T., Alsowail, R. A., & Pandiaraj, S. (2023). Insider threat detection model using anomaly-based isolation forest algorithm. *IEEE Access*, *11*, 118170–118185. https://doi.org/10.1109/ACCESS.2023.3326750

Al-Shehari, T., & Alsowail, R. A. (2023). Random resampling algorithms for addressing the imbalanced dataset classes in insider threat detection. *International Journal of Information Security*, *22*(3), 611–629.

Alshehri, A. (2022). Relational deep learning detection with multi-sequence representation for insider threats. *International Journal of Advanced Computer Science and Applications*, *13*, 758–765. http://dx.doi.org/10.14569/IJACSA.2022.0130587

AlSlaiman, M., Salman, M. I., Saleh, M. M., & Wang, B. (2023). Enhancing false negative and positive rates for efficient insider threat detection. *Computers and Security*, *126*. http://dx.doi.org/10.1016/j.cose.2022.103066

Alsowail, R. A., & Al-Shehari, T. (2020). Empirical detection techniques of insider threat incidents. *IEEE Access*, *8*, 78385–78402. http://dx.doi.org/10.1109/ACCESS.2020.2989739

Amer, M., & Abdennadher, S. (2011). Comparison of unsupervised anomaly detection techniques. *Bachelor's Thesis*.

Anju, A., & Krishnamurthy, M. (2024). M-eos: Modified-equilibrium optimization-based stacked cnn for insider threat detection. *Wireless Networks*, *30*(4), 2819–2838. http://dx.doi.org/10.1007/s11276-024-03678-5

Singh, M., Sangeetha, S., & Mehtre, B. (2023). Insider threat detection and prevention using semantic score and dynamic multi-fuzzy classifier. *International Journal of Ad Hoc and Ubiquitous Computing*, *42*(2), 95–112. http://dx.doi.org/10.1504/IJAHUC.2022.10046481

Asha, S., D, S., & G, P. (2023). Malicious insider threat detection using variation of sampling methods for anomaly detection in cloud environment. *Computers and Electrical Engineering*, *105*. http://dx.doi.org/10.1016/j.compeleceng.2022.108519

Bapna, A., Arivazhagan, N., & Firat, O. (2019). Simple, scalable adaptation for neural machine translation. *arXiv preprint arXiv:1909.08478*.

Bartoszewski, F. W. (2022). *Machine learning and anomaly detection for insider threat detection* [Doctoral dissertation, Heriot-Watt University].

Bartoszewski, F. W., Just, M., Lones, M. A., & Mandrychenko, O. (2021). Anomaly detection for insider threats: An objective comparison of machine learning models and ensembles. *IFIP Advances in Information and Communication Technology*, *625*, 367–381. http://dx.doi.org/10.1007/978-3-030-78120-0_24

Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). Lof: Identifying density-based local outliers. *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 93–104.

Cai, X., Wang, Y., Xu, S., Li, H., Zhang, Y., & Yuan, X. (2024). Lan: Learning adaptive neighbors for real-time insider threat detection. *arXiv preprint arXiv:2403.09209*.

Cervellera, C., & Macciò, D. (2017). Distribution-preserving stratified sampling for learning problems. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(7), 2886–2895.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, *41*(3), 1–58.

Chattopadhyay, P., Wang, L., & Tan, Y.-P. (2018). Scenario-based insider threat detection from cyber activities. *IEEE Transactions on Compu-*

tational Social Systems, *5*(3), 660–675. http://dx.doi.org/10.1109/TCSS.2018.2857473

Chen, Nyemba, S., & Malin, B. (2012). Detecting anomalous insiders in collaborative information systems. *IEEE Transactions on Dependable and Secure Computing*, *9*(3), 332–344. http://dx.doi.org/10.1109/TDSC.2012.11

Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Chollet, F. (2021). *Deep learning with python*. Simon; Schuster.

chung Liu, A. Y. (2004). The effect of oversampling and undersampling on classifying imbalanced text datasets.

CISA - Defining Insider Threats. (2023). Defining insider threats [Accessed: December 15, 2023].

CISA - Insider Threat Mitigation Guide. (2020). *Cisa - insider threat mitigation guide*. Cybersecurity and Infrastructure Security Agency. Washington, DC, USA. https://www.cisa.gov/sites/default/files/2022-11/Insider%20Threat%20Mitigation%20Guide_Final_508.pdf

Devince, H. C. (2020). Understanding controls to detect and mitigate malicious privileged user abuse.

Dongyang, Yang, L., Zhang, H., Wang, X., & Ma, L. (2022). Memory-augmented insider threat detection with temporal-spatial fusion. *Security and Communication Networks*, *2022*. http://dx.doi.org/10.1155/2022/6418420

Dongyang, Yang, L., Zhang, H., Wang, X., Ma, L., & Xiao, J. (2021). Image-based insider threat detection via geometric transformation. *Security and Communication Networks*, *2021*. http://dx.doi.org/10.1155/2021/1777536

Emmott, A., Das, S., Dietterich, T., Fern, A., & Wong, W.-K. (2015). A meta-analysis of the anomaly detection problem. *arXiv preprint arXiv:1503.01158*.

Gayathri, R., Sajjanhar, A., & Xiang, Y. (2020). Image-based feature representation for insider threat classification. *Applied Sciences*, *10*(14), 4945.

Gill, P. (2012). Intelligence, threat, risk and the challenge of oversight. *Intelligence and National Security*, *27*(2), 206–222.

Glasser, J., & Lindauer, B. (2013). Bridging the gap: A pragmatic approach to generating insider threat data. *Proceedings - IEEE CS Security and Privacy Workshops, SPW 2013*, 98–104. http://dx.doi.org/10.1109/SPW.2013.37

Gogoi, P., Bhattacharyya, D. K., Borah, B., & Kalita, J. K. (2011). A survey of outlier detection methods in network anomaly identification. *The Computer Journal*, *54*(4), 570–588.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Greitzer, F. L., Frincke, D., & Zabriskie, M. (2011). Social/ethical issues in predictive insider threat monitoring. In *Information assurance and security ethics in complex systems: Interdisciplinary perspectives* (pp. 132–161). IGI Global.

Greitzer, F. L., & Hohimer, R. E. (2011). Modeling human behavior to anticipate insider attacks. *Journal of Strategic Security*, *4*, 25–48. https://doi.org/10.5038/1944-0472.4.2.2

Greitzer, F. L., Purl, J., Leong, Y. M., & Sticha, P. J. (2019). Positioning your organization to respond to insider threats. *IEEE Engineering Management Review*, *47*(2), 75–83. https://doi.org/10.1109/EMR.2019.2914612

Gruetzemacher, R., & Paradice, D. (2022). Deep transfer learning & beyond: Transformer language models in information systems research. *ACM Computing Surveys (CSUR)*, *54*(10s), 1–35.

Guo, H., Lin, X., Yang, J., Zhuang, Y., Bai, J., Zhang, B., Zheng, T., & Li, Z. (2021). *Translog: A unified transformer-based framework for log anomaly detection*.

Gurucul. (2023a). https://gurucul.com/2023-insider-threat-report?qgad=678205213083&qgterm=insider%20threat%20report&utm_source=google&utm_medium=cpc&utm_campaign=consideration&utm_adgroup=%7Badgroup%7D&utm_term=insider%20threat%20report&gad_source=1&gclid=CjwKCAiApuCrBhAuEiwA8VJ6JnSzSdy-OdpWmzMvJ-yBf9NN-S7iZfNtNAB28Ok3yQoCvyhGTQ8vjRoCqPYQAvD_BwE

Gurucul. (2023b, August). Famous insider threat cases. https://gurucul.com/blog/famous-insider-threat-cases

Gyanchandani, M., Rana, J., & Yadav, R. (2012). Taxonomy of anomaly based intrusion detection system: A review. *International Journal of Scientific and Research Publications*, *2*(12), 1–13.

Haidar, D., & Gaber, M. M. (2018). Adaptive one-class ensemble-based anomaly detection: An application to insider threats. *Proceedings of the International Joint Conference on Neural Networks*, *2018-July*. http://dx.doi.org/10.1109/IJCNN.2018.8489107

Hanley, M., & Montelibano, J. (2011). *Insider threat control: Using centralized logging to detect data exfiltration near insider termination, technical note, cert* (tech. rep.). CMU/SEI-2011-TN-024, Carnegie Mellon University. Software Engineering Institute.

Heejung, K., & Hwankuk, K. (2022). Comparative experiment on ttp classification with class imbalance using oversampling from cti dataset. *Security and Communication Networks*, *2022*. http://dx.doi.org/10.1155/2022/5021125

Hoang, T. L., Pham, T. D., & Ta, V. C. (2021). Improving graph convolutional networks with transformer layer in social-based items recommendation. *2021 13th International Conference on Knowledge and Systems Engineering (KSE)*, 1–6.

Homoliak, I., Toffalini, F., Guarnizo, J., Elovici, Y., & Ochoa, M. (2019). Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures. *ACM Computing Surveys (CSUR)*, *52*(2), 1–40.

Hong, W., Yin, J., You, M., Wang, H., Cao, J., Li, J., Liu, M., & Man, C. (2023). A graph empowered insider threat detection framework based on daily activities. *ISA transactions*, *141*, 84–92.

Hossin, M., & Sulaiman, M. N. (2015). A review on evaluation metrics for data classification evaluations. *International journal of data mining & knowledge management process*, *5*(2), 1.

Huang, W., Zhu, H., Li, C., Lv, Q., Wang, Y., & Yang, H. (2021). Itdbert: Temporal-semantic representation for insider threat detection. *2021 IEEE Symposium on Computers and Communications (ISCC)*, 1–7.

IBM Security. (2023). *Cost of a data breach report 2023* (tech. rep.) (Accessed: 2024-03-27). IBM. https://www.ibm.com/reports/data-breach

Jang, M., Ryu, Y., Kim, J.-S., & Cho, M. (2020a). Against insider threats with hybrid anomaly detection with local-feature autoencoder and global statistics (lags). *IEICE Transactions on Information and Sys-*

tems, *E103D*(4), 888–891. http://dx.doi.org/10.1587/transinf.2019EDL8180

Jang, M., Ryu, Y., Kim, J.-S., & Cho, M. (2020b). Against insider threats with hybrid anomaly detection with local-feature autoencoder and global statistics (lags). *IEICE Transactions on Information and Systems*, *E103.D*, 888–891. https://doi.org/10.1587/transinf.2019EDL8180

Janjua, F., Masood, A., Abbas, H., Rashid, I., & Khan, M. M. Z. M. (2021). Textual analysis of traitor-based dataset through semi supervised machine learning. *Future Generation Computer Systems*, *125*, 652–660. http://dx.doi.org/10.1016/j.future.2021.06.036

Kabeireho, M. (2020). Impact of deterrence and user awareness on insider threat vulnerability: A correlational study. *ProQuest Dissertations and Theses Global*.

Kanter, J. M., & Veeramachaneni, K. (2015). Deep feature synthesis: Towards automating data science endeavors. *2015 IEEE international conference on data science and advanced analytics (DSAA)*, 1–10.

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., & Shah, M. (2022). Transformers in vision: A survey. *ACM Computing Surveys*, *54*(10). http://dx.doi.org/10.1145/3505244

Kim, Kim, C. M., & Yim, M.-S. (2020). An investigation of insider threat mitigation based on eeg signal classification. *Sensors (Switzerland)*, *20*(21), 1–17. http://dx.doi.org/10.3390/s20216365

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kolb, S., Paramonov, S., Guns, T., & De Raedt, L. (2017). Learning constraints in spreadsheets and tabular data. *Machine Learning*, *106*, 1441–1468.

Le, D. C., & Zincir-Heywood, N. (2021a). Anomaly detection for insider threats using unsupervised ensembles. *IEEE Transactions on Network and Service Management*, *18*, 1152–1164. http://dx.doi.org/10.1109/TNSM.2021.3071928

Le, D. C., & Zincir-Heywood, N. (2021b). Exploring anomalous behaviour detection and classification for insider threat identification. *International Journal of Network Management*, *31*(4), e2109.

Lee, Y., & Kang, P. (2022). Anovit: Unsupervised anomaly detection and localization with vision transformer-based encoder-decoder [Anomaly detection;Anomaly localizations;Features extraction;Images reconstruction;Location awareness;MVTecAD;Task analysis;Transformer;Vision transformer;]. *IEEE Access*, *10*, 46717–46724. http://dx.doi.org/10.1109/ACCESS.2022.3171559

Legg, P. A., Buckley, O., Goldsmith, M., & Creese, S. (2017). Automated insider threat detection system using user and role-based profile assessment. *IEEE Systems Journal*, *11*(2), 503–512. http://dx.doi.org/10.1109/JSYST.2015.2438442

Legg, P. A., Moffat, N., Nurse, J. R., Happa, J., Agrafiotis, I., Goldsmith, M., & Creese, S. (2013). Towards a conceptual model and reasoning structure for insider threat detection. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, *4*, 20–37.

Li, Chen, Z., & Zheng, J. (2022). An efficient transformer encoder-based classification of malware using api calls. *Proceedings - 24th IEEE International Conference on High Performance Computing and Communications, 8th IEEE International Conference on Data Science and Systems, 20th IEEE International Conference on Smart City and 8th IEEE International Conference on Dependability in Sensor, Cloud and*

Big Data Systems and Application, HPCC/DSS/SmartCity/DependSys 2022, 839–846. http://dx.doi.org/10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00137

Lindauer, B. (2020). *Insider threat test dataset* (Version v1). Carnegie Mellon University. https://doi.org/10.1184/R1/12841247.v1

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation forest. *2008 eighth ieee international conference on data mining*, 413–422.

Liu, L., De Vel, O., Han, Q.-L., Zhang, J., & Xiang, Y. (2018). Detecting and preventing cyber insider threats: A survey. *IEEE Communications Surveys and Tutorials*, *20*(2), 1397–1418. http://dx.doi.org/10.1109/COMST.2018.2800740

Lo, O., Buchanan, W. J., Griffiths, P., & Macfarlane, R. (2018). Distance measurement methods for improved insider threat detection. *Security and Communication Networks*, *2018*, 1–18.

Maasberg, M., Zhang, X., Ko, M., Miller, S. R., & Beebe, N. L. (2020a). An analysis of motive and observable behavioral indicators associated with insider cyber-sabotage and other attacks. *IEEE Engineering Management Review*, *48*, 151–165. http://dx.doi.org/10.1109/EMR.2020.2989108

Maasberg, M., Zhang, X., Ko, M., Miller, S. R., & Beebe, N. L. (2020b). An analysis of motive and observable behavioral indicators associated with insider cyber-sabotage and other attacks. *IEEE Engineering Management Review*, *48*(2), 151–165. http://dx.doi.org/10.1109/EMR.2020.2989108

Le, D. C., Zincir-Heywood, N., & Heywood, M. I. (2020). Analyzing data granularity levels for insider threat detection using machine learning.

IEEE Transactions on Network and Service Management, *17*(1), 30–44. http://dx.doi.org/10.1109/TNSM.2020.2967721

Mehmood, M., Amin, R., Muslam, M. M. A., Xie, J., & Aldabbas, H. (2023). Privilege escalation attack detection and mitigation in cloud using machine learning. *IEEE Access*, *11*, 46561–46576. http://dx.doi.org/10.1109/ACCESS.2023.3273895

Milosevic, M. S., & Ciric, V. M. (2022). Extreme minority class detection in imbalanced data for network intrusion. *Computers and Security*, *123*. http://dx.doi.org/10.1016/j.cose.2022.102940

Naseem, U., Razzak, I., Musial, K., & Imran, M. (2020). Transformer based deep intelligent contextual embedding for twitter sentiment analysis. *Future Generation Computer Systems*, *113*, 58–69.

Nasir, R., Afzal, M., Latif, R., & Iqbal, W. (2021). Behavioral based insider threat detection using deep learning. *IEEE Access*, *9*, 143266–143274.

Nasser, A.-M., Ahmad, R., Abidin, Z. Z., Abdulkareem, K. H., Mohammed, M. A., Gupta, D., & Shankar, K. (2022). A new intelligent multilayer framework for insider threat detection. *Computers and Electrical Engineering*, *97*.

National Insider Threat Task Force. (2017). *National insider threat task force (nittf) insider threat guide* [1/24/2024]]. https://www.dni.gov/files/NCSC/documents/nittf/NITTF-Insider-Threat-Guide-2017.pdf

Nguyen, N., Reiher, P., & Kuenning, G. H. (2003). Detecting insider threats by monitoring system call activity. *IEEE Systems, Man and Cybernetics SocietyInformation Assurance Workshop, 2003.*, 45–52.

Novak, N. (2019). Python dictionary.

Nurse, J. R., Buckley, O., Legg, P. A., Goldsmith, M., Creese, S., Wright, G. R., & Whitty, M. (2014). Understanding insider threat: A framework

for characterising attacks. *Proceedings - IEEE Symposium on Security and Privacy*, *2014-January*, 214–228. http://dx.doi.org/10.1109/SPW.2014.38

Olano, M., Sherman, A., Oliva, L., Cox, R., Firestone, D., Kubik, O., Patil, M., Seymour, J., Sohn, I., & Thomas, D. (2014). Securityempire: Development and evaluation of a digital game to promote cybersecurity education. *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education, 3GSE 2014*, NSF–.

Orabi, M. (2023). *Enable transformers for anomaly detection in multivariate time series data* [Retrieved February 23, 2024]. https://medium.com/@moussab.orabi/enable-transformers-with-anomaly-detection-in-high-order-multivariate-time-series-data-509a5df39151

Padayachee, K. (2016). An assessment of opportunity-reducing techniques in information security: An insider threat perspective. *Decision Support Systems*, *92*, 47–56. http://dx.doi.org/10.1016/j.dss.2016.09.012

Pal, P., Chattopadhyay, P., & Swarnkar, M. (2023). Temporal feature aggregation with attention for insider threat detection from activity logs. *Expert Systems with Applications*, *224*, 119925. https://doi.org/10.1016/j.eswa.2023.119925

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32*.

Pembury Smith, M. Q., & Ruxton, G. D. (2020). Effective use of the mcnemar test. *Behavioral Ecology and Sociobiology*, *74*, 1–9.

Ponemon. (2023). Cost of insider risks [Independently conducted by Ponemon Institute].

Prakash, A., Chitta, K., & Geiger, A. (2021). Multi-modal fusion transformer for end-to-end autonomous driving. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7077–7087.

Preeti, Bala, R., & Singh, R. P. (2022). A dual-stage advanced deep learning algorithm for long-term and long-sequence prediction for multivariate financial time series. *Applied Soft Computing*, *126*. http://dx.doi.org/10.1016/j.asoc.2022.109317

Randive, K., Mohan, R., & Sivakrishna, A. M. (2023). An efficient pattern-based approach for insider threat classification using the image-based feature representation. *Journal of Information Security and Applications*, *73*. http://dx.doi.org/10.1016/j.jisa.2023.103434

Randive, K., & Ramasundaram, M. (2023). Mwcapsnet: A novel multi-level wavelet capsule network for insider threat detection using image representations. *Neurocomputing*, *553*. http://dx.doi.org/10.1016/j.neucom.2023.126588

Raschka, S. (2018). Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*.

Safa, N. S., Maple, C., Watson, T., & Von Solms, R. (2018). Motivation and opportunity based model to reduce information security insider threats in organisations. *Journal of Information Security and Applications*, *40*, 247–257. http://dx.doi.org/10.1016/j.jisa.2017.11.001

Sagvekar, V. R., & Sharma, P. (2023). Word embedding attention and balanced cross entropy technique for sentiment analysis. *Multiagent and Grid Systems*, *19*(1), 23–42. http://dx.doi.org/10.3233/MGS-221511

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural computation*, *13*(7), 1443–1471.

Schwertner, K. (2017). Digital transformation of business. *Trakia Journal of Sciences*, *15*(1), 388–393.

Senator, T. E., Goldberg, H. G., Memory, A., Young, W. T., & Rees, B. (2013). Detecting insider threats in a real corporate database of computer usage activity. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *Part F128815*, 1393–1401. http://dx.doi.org/10.1145/2487575.2488213

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, *404*, 132306.

Smith, K. (2018). Using the service desk to secure the organisation. *Computer Fraud and Security*, *2018*(1), 11–14. http://dx.doi.org/10.1016/S1361-3723(18)30006-X

Software Engineering Institute. (2022, September). *Common sense guide to mitigating insider threats, seventh edition* (tech. rep.) (Accessed: 2024-Jan-24). https://insights.sei.cmu.edu/library/common-sense-guide-to-mitigating-insider-threats-seventh-edition/

Soh, C., Yu, S., Narayanan, A., Duraisamy, S., & Chen, L. (2019). Employee profiling via aspect-based sentiment and network for insider threats detection. *Expert Systems with Applications*, *135*, 351–361. http://dx.doi.org/10.1016/j.eswa.2019.05.043

Song, S., Gao, N., Zhang, Y., & Ma, C. (2024). Britd: Behavior rhythm insider threat detection with time awareness and user adaptation. *Cybersecurity*, *7*(1). http://dx.doi.org/10.1186/s42400-023-00190-9

Qiang, L., Li, P., Zhao, W., Cai, W., Yu, S., & Leung, V. C. M. (2018). A survey on security threats and defensive techniques of machine learning:

A data driven view [Automatic driving;defensive techniques;Facial recognition;Malware detection;NAtural language processing;Principle component analysis;Security assessment;Security threats;]. *IEEE Access*, *6*, 12103–12117. http://dx.doi.org/10.1109/ACCESS.2018.2805680

Tae-Young, K., & Cho, S.-B. (2018). Web traffic anomaly detection using c-lstm neural networks. *Expert Systems with Applications*, *106*, 66–76. http://dx.doi.org/10.1016/j.eswa.2018.04.004

Taormina, R., & Galelli, S. (2018). Deep-learning approach to the detection and localization of cyber-physical attacks on water distribution systems. *Journal of Water Resources Planning and Management*, *144*(10), 04018065.

Tuli, S., Casale, G., & Jennings, N. R. (2022). *Tranad: Deep transformer networks for anomaly detection in multivariate time series data*. http://dx.doi.org/10.48550/arXiv.2201.07284

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Villarreal-Vasquez, M., Modelo-Howard, G., Dube, S., & Bhargava, B. (2023). Hunting for insider threats using lstm-based anomaly detection. *IEEE Transactions on Dependable and Secure Computing*, *20*(1), 451–462. http://dx.doi.org/10.1109/TDSC.2021.3135639

Wang, Cao, Y., & An, H. (2022). Gray-code fringe order jump error self-correction based on shifted phase encoding for phase measuring profilometry. *Optics Communications*, *524*, 128763.

Wang & El Saddik, A. (2023). Dtitd: An intelligent insider threat detection framework based on digital twin and self-attention based deep learn-

ing models. *IEEE Access*, *11*, 114013–114030. http://dx.doi.org/10.1109/ACCESS.2023.3324371

Wei, Y., Chow, K.-P., & Yiu, S.-M. (2021). Insider threat prediction based on unsupervised anomaly detection scheme for proactive forensic investigation. *Forensic Science International: Digital Investigation*, *38*. http://dx.doi.org/10.1016/j.fsidi.2021.301126

Williams, A. D., Abbott, S. N., Shoman, N., & Charlton, W. S. (2022). Results from invoking artificial neural networks to measure insider threat detection and mitigation. *Digital Threats: Research and Practice*, *3*(1). http://dx.doi.org/10.1145/3457909

Wu, Z., Zhang, H., Wang, P., & Sun, Z. (2022). Rtids: A robust transformer-based approach for intrusion detection system. *IEEE Access*, *10*, 64375–64387.

Xiao, J., Yang, L., Zhong, F., Wang, X., Chen, H., & Li, D. (2023). Robust anomaly-based insider threat detection using graph neural network. *IEEE Transactions on Network and Service Management*, *20*, 3717–3733. http://dx.doi.org/10.1109/TNSM.2022.3222635

Yao, D. D., Shu, X., Cheng, L., & Stolfo, S. J. (2018). Anomaly detection as a service: Challenges, advances, and opportunities. *Synthesis Lectures on Information Security, Privacy, and Trust*, *9*, 1–175. http://dx.doi.org/10.2200/S00800ED1V01Y201709SPT022

Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, *5*, 21954–21961. http://dx.doi.org/10.1109/ACCESS.2017.2762418

Yu, J., Zheng, W., Chen, Y., Zhang, Y., & Huang, R. (2023). Surrounding-aware representation prediction in birds-eye-view using transformers. *Frontiers in Neuroscience*, *17*.

Yuan, S., & Wu, X. (2021). Deep learning for insider threat detection: Review, challenges and opportunities. *Computers and Security*, *104*. http://dx.doi.org/10.1016/j.cose.2021.102221

Zhang, J., Chen, Y., & Ju, A. (2018). Insider threat detection of adaptive optimization dbn for behavior logs. *Turkish Journal of Electrical Engineering and Computer Sciences*, *26*(2), 792–802. http://dx.doi.org/10.3906/elk-1706-163

Zhao, Z., & Sun, B. (2023). Hyperspectral anomaly detection via memory-augmented autoencoders [Abnormal samples;Auto encoders;Background reconstruction;Hyperspectral anomaly detection;Hyperspectral Data;Low dimensional;matrix;Memory modules;Performance;Reconstruction error;]. *CAAI Transactions on Intelligence Technology*, *8*(4), 1274–1287. http://dx.doi.org/10.1049/cit2.12116

Zhenjiang, Z., Zhao, L., Zhang, Y., Zhou, H., & Li, W. (2021). Research on insider threat detection method based on variational autoencoding. *Journal of Computers*, *32*(4), 201–210.

Zou, S., Sun, H., Xu, G., & Quan, R. (2020). Ensemble strategy for insider threat detection from user activity logs. *Computers, Materials and Continua*, *65*(2), 1321–1334. http://dx.doi.org/10.32604/cmc.2020.09649

Appendix A:

A.1 Adaptive Malicious Events Learners

A.1.1 Transformer Training Code

Listing A.1 provides a summary of the code —Adaptive Malicious Events Learner— used to construct, train, and evaluate the Transformer encoder.

```
#!/usr/bin/env python
# Author : MOHAMED ELBASHEER

# ----- 1. Import the modules -----#
import torch
import numpy as np
import pandas as pd
import os
import math
import torch.nn.functional as F
from torch.utils.tensorboard import SummaryWriter
from datetime import datetime
from sklearn.metrics import (
    precision_recall_fscore_support,
    confusion_matrix,
    classification_report,
    accuracy_score,
)
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader, TensorDataset
from torchsummaryX import summary
from torchinfo import summary
import torch.nn.init as init
from datetime import timedelta
from statsmodels import robust
import matplotlib.pyplot as plt
from statsmodels.robust import mad as robust_mad
from torchviz import make_dot
import hiddenlayer as hl
import pickle
os.environ["PATH"] += os.pathsep + "C:/Program Files/Graphviz/bin/"
import time
import random
import string
import warnings

warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
start_time = datetime.now()

# ----- 2. Set the Constants  -----#
D_MODEL = 320
NHEAD = 8
NUM_ENCODER_LAYERS = 6
DIM_FEEDFORWARD = 512
INPUT_DIM = 9 * 35  # 9 * 35 = 315
BATCH_SIZE = 1
NUM_EPOCHS = 57
PRINT_EVERY = 100
LR = 0.00001

# ----- 3. Set the Directories       -#
```

```
DATA_DIRECTORY = "C:\\PRAXIS-FINAL\\TRANSFORMERS\\datasets\\"
MODEL_DIRECTORY = "C:\\PRAXIS-FINAL\\TRANSFORMERS\\train_model\\"
ARCHITECTURE_DIRECTORY = "C:\\PRAXIS-FINAL\\TRANSFORMERS\\architecture\\"
OUTPUT_DIRECTORY = "C:\\PRAXIS-FINAL\\TRANSFORMERS\\log\\"
TENSOR_LOG = "C:\\PRAXIS-FINAL\\TRANSFORMERS\\log\\"
MINDEX = "C:\PRAXIS-FINAL\\TRANSFORMERS\\multi_index_dataset\\"

# ----- 4. Set Training and Validation files #
# Training and Validation files
TRAINING_FILE = "benign_train_700.pt"
VALIDATION_FILE = "benign_val_270.pt"
TRAINING_SET = DATA_DIRECTORY + TRAINING_FILE
VALIDATION_SET = DATA_DIRECTORY + VALIDATION_FILE

# ----- 5. Utility Functions  -----#
def weights_init_xavier(m):
    if isinstance(m, nn.Linear):
        torch.nn.init.xavier_uniform_(m.weight)
        torch.nn.init.zeros_(m.bias)

# ----- 6. Training Data loader----#
def train_data():
    user_data_train = torch.load(TRAINING_SET)
    train_data_list = []
    for user_id, tensor in user_data_train.items():
        train_data_list.append(tensor)

    train_dataset = [(tensor,) for tensor in train_data_list]
    train_loader = DataLoader(
        dataset=train_dataset, batch_size=BATCH_SIZE, shuffle=True
    )
    return train_loader


def val_data():
    user_data_test = torch.load(VALIDATION_SET)
    test_data_list = [tensor for user_id, tensor in user_data_test.items()]
    test_dataset = [(tensor,) for tensor in test_data_list]
    test_loader = DataLoader(dataset=test_dataset, batch_size=1, shuffle=False)
    return test_loader

# ----- 7. Model Definition    ----#
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(
            torch.arange(0, d_model, 2).float() * (-math.log(10000.0) / d_model)
        )
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer("pe", pe)

    def forward(self, x):
        x = x + self.pe[: x.size(0), :, : x.size(2)]
        return self.dropout(x)


class TransformerEncoderModel(nn.Module):
    def __init__(
        self,
        input_dim,
        d_model,
        nhead,
        num_encoder_layers,
        dim_feedforward,
        dropout=0.1,
    ):
```

```
        super(TransformerEncoderModel, self).__init__()
        self.input_projection = nn.Linear(input_dim, d_model)
        self.pos_encoder = PositionalEncoding(d_model, dropout)
        self.transformer_encoder_layer = nn.TransformerEncoderLayer(
         d_model, nhead, dim_feedforward, dropout=dropout)
        self.transformer_encoder = nn.TransformerEncoder(
        self.transformer_encoder_layer, num_encoder_layers)
        self.output_projection = nn.Linear(
            d_model, input_dim
        )  # Project back to original dimension
        self.layer_norm = nn.LayerNorm(d_model)

    def forward(self, src):
        src = src.view(src.size(0), src.size(1), -1).transpose(0, 1)
        src = self.input_projection(src)
        src = self.pos_encoder(src)
        src = self.transformer_encoder(src)
        src = self.layer_norm(src)
        output = self.output_projection(src)
        output = output.transpose(0, 1).view(output.size(1), output.size(0), 9, 35)
        return output

# ----- 8. Training Loop    -------#
def train_model(model, train_loader, val_loader, history):
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=LR)
    writer = SummaryWriter(log_dir=TENSOR_LOG)
    training_losses = []
    validation_losses = []
    all_errors = []  # Initialize outside the loop
    for epoch in range(NUM_EPOCHS):
        model.train()  # Set the model to training mode
        total_train_loss = 0.0
        # Training loop
        for batch_idx, inputs in enumerate(train_loader):
            inputs = inputs[0].to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, inputs)
            loss.backward()
            optimizer.step()
            total_train_loss += loss.item()
        average_train_loss = total_train_loss / len(train_loader)
        training_losses.append(average_train_loss)
        writer.add_scalar("Loss/train", average_train_loss, epoch)
        history.log(epoch, train_loss=average_train_loss)

        # Evaluation loop
        model.eval()  # Set the model to evaluation mode
        total_val_loss = 0.0
        if epoch == NUM_EPOCHS - 1:  # Only compute errors for the last epoch
            print(f"This is the last epoch {epoch}")
            all_errors = []  # Reset to store errors only for the last epoch
        with torch.no_grad():
            for inputs in val_loader:
                inputs = inputs[0].to(device)
                outputs = model(inputs)
                loss = criterion(outputs, inputs)
                total_val_loss += loss.item()
                if (
                    epoch == NUM_EPOCHS - 1
                ):  # Compute and store errors only for the last epoch
                    error = torch.abs(outputs - inputs)
                    median_error = error.median(dim=1)[0].median(dim=1)[0]
                    all_errors.extend(median_error.cpu().numpy())

        average_val_loss = total_val_loss / len(val_loader)
        validation_losses.append(average_val_loss)
        writer.add_scalar("Loss/val", average_val_loss, epoch)
        history.log(epoch, val_loss=average_val_loss)

```

```
        print(
            f"Epoch {epoch}/{NUM_EPOCHS}, Average Train Loss: {average_train_loss:.4f}, Average Val Loss: {average_val_loss:.4f}"
        )

    writer.close()
    return model, all_errors, training_losses, validation_losses

# ----- 9. Main Execution    ------#
starttime = time.perf_counter()
history = hl.History()
train_loader = train_data()
val_loader = val_data()
model = TransformerEncoderModel(
    9 * 35, D_MODEL, NHEAD, NUM_ENCODER_LAYERS, DIM_FEEDFORWARD, dropout=0.1)
model.apply(weights_init_xavier)
model = model.to(device)

# ----10. Tensorboard Computational graph #
mysequence_length = 500
mybatch_size = 1
myinput_features = 9 * 35
dummy_input = torch.randn(mysequence_length, mybatch_size, myinput_features)
# To ensure the datatype match my model
dummy_input = dummy_input.float().to(device)
# initialize the SummaryWriter
writer = SummaryWriter(log_dir=TENSOR_LOG)
# Then add the model graph to the summary
with writer:
    writer.add_graph(model, dummy_input)

# ---11. Print model summary  #
input_shape = (1, 501, 9 * 35)  # (batch_size, sequence_length, feature_dimension)
model_summary = summary(model, input_size=input_shape)
print(model_summary)

# ---12. create Graphviz PNG  #
# create a dummy input for the model where it shape will match my input size.
dummy_input = torch.randn(input_shape)
model = model.to(device)
dummy_input = dummy_input.to(device)
# then create the Graphviz dot file
dot = make_dot(model(dummy_input), params=dict(model.named_parameters()))
# in order to visualize it in Jupyter Notebook
dot.format = "png"
architecture_file_name = (
    f"Architecture_Epochs_{NUM_EPOCHS}_Lr_{LR}_Time_{timestamp}.png")
# dot.render("model_architecture")
dot.render(filename=architecture_file_name, directory=ARCHITECTURE_DIRECTORY)
history = hl.History()
canvas = hl.Canvas()

# ---13. Model Training   #
model, errors_production, training_losses, validation_losses = train_model(
    model, train_loader, val_loader, history)
# Plotting the training and validation losses
plt.plot(training_losses, label="Training Loss")
plt.plot(validation_losses, label="Validation Loss")
plt.title("Training vs Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

# Set the filename to save the losses in
losses_filename = (
    f"{OUTPUT_DIRECTORY}Losses_Epochs_{NUM_EPOCHS}_Lr_{LR}_Time_{timestamp}.pkl")
# Save the history to a file.  This is for hidden layer
with open(losses_filename, "wb") as f:
    pickle.dump(history.history, f)
errors_array = np.array(errors_production)
```

```
errors_df = pd.DataFrame(
    errors_array, columns=[f"Feature_{i}" for i in range(errors_array.shape[1])])

reconstruction_file_name = (
    f"{OUTPUT_DIRECTORY}Errors_Epochs_{NUM_EPOCHS}_Lr_{LR}_Time_{timestamp}.pkl")
# Save the DataFrame to a pickle file
errors_df.to_pickle(reconstruction_file_name)

# --14. save the Model     -#
model_filename = (
    f"{MODEL_DIRECTORY}Model_Epochs_{NUM_EPOCHS}_Lr_{LR}_Time_{timestamp}.pth")
torch.save(model.state_dict(), model_filename)
print(f"MODEL.................:\n  {model_filename}")
print(f"ERROR.................:\n  {reconstruction_file_name}")
print(f"LOSSES................:\n  {losses_filename}")
print(f"ARCHITECTURE..........:\n  {ARCHITECTURE_DIRECTORY+architecture_file_name}")
end_time = datetime.now()
print("\n")
print(f"It took {end_time - start_time} to finish running the training/Evaluation.")
import pickle
import plotly.graph_objects as go
# Load the pickle file
with open(losses_filename, "rb") as f:
    loaded_logs = pickle.load(f)
# Extract training and validation losses
train_losses = [data["train_loss"] for data in loaded_logs.values()]
val_losses = [data["val_loss"] for data in loaded_logs.values()]
# Create traces
fig = go.Figure()
fig.add_trace(go.Scatter(y=train_losses, mode="lines", name="Training Loss"))
fig.add_trace(go.Scatter(y=val_losses, mode="lines", name="Validation Loss"))
# Edit the layout
fig.update_layout(
    title="Training and Validation Losses", xaxis_title="Epoch", yaxis_title="Loss")
# Show plot
fig.show()
import pickle
import plotly.graph_objects as go
losses_filename = "C:\\PRAXIS-FINAL\\TRANSFORMERS\\log\\Losses_Epochs_57_Lr_1e-05
    _Time_2023-11-20_12-42-46.pkl"
# Load the pickle file
with open(losses_filename, "rb") as f:
    loaded_logs = pickle.load(f)
# Extract training and validation losses
train_losses = [data["train_loss"] for data in loaded_logs.values()]
val_losses = [data["val_loss"] for data in loaded_logs.values()]
# Create traces
fig = go.Figure()
fig.add_trace(go.Scatter(y=train_losses, mode="lines", name="Training Loss"))
fig.add_trace(go.Scatter(y=val_losses, mode="lines", name="Validation Loss"))
# Edit the layout
fig.update_layout(
    title="Training and Validation Losses", xaxis_title="Epoch", yaxis_title="Loss")
fig.update_layout(width=1000, height=600)
# Show plot
fig.show()
with open(reconstruction_file_name, "rb") as f:
    loaded_errors = pickle.load(f)
loaded_errors.head(5)
# An estimate of feature importance  (based on the reconstruction error)
feature_importance = loaded_errors.mean().sort_values()
feature_importance.plot(kind="barh", figsize=(10, 8))
plt.title("Average Reconstruction Error for Each Feature")
plt.xlabel("Average Reconstruction Error")
plt.ylabel("Feature")
plt.show()
#  A histogram for each feature to visualize the distribution of reconstruction
    errors.
loaded_errors.hist(bins=50, figsize=(15, 15))
plt.suptitle("Distribution of Reconstruction Errors for Each Feature")
plt.show()
```

```
# A heatmap to visualize the reconstruction errors and the original data the errors
    are based on
original_data = pd.read_pickle(MINDEX + "benign_val_270.pkl")
# Now plot the heatmap for the Reconstruction Errors as well as the original data (
    side by side)
import matplotlib.pyplot as plt
import seaborn as sns
fig, axes = plt.subplots(ncols=2, figsize=(12, 6))
sns.heatmap(loaded_errors, cmap="viridis", cbar=False, ax=axes[0])
axes[0].set_title("Heatmap of Reconstruction Errors")
axes[0].set_xlabel("Feature")
axes[0].set_ylabel("Instance")
sns.heatmap(original_data, cmap="viridis", cbar=False, ax=axes[1])
axes[1].set_title("Heatmap of Original Data")
axes[1].set_xlabel("Feature")
axes[1].set_ylabel("Instance")
# To turn off features ticks if its too crowded. Otherwise comment them
axes[0].set_xticklabels([])
axes[1].set_xticklabels([])
axes[0].set_yticklabels([])
axes[1].set_yticklabels([])
plt.show()
```

Source Code A.1: Transformer Training

A.1.2 Transformer Testing Code

Listing A.2 provides a summary of the code used to load and test the Transformer encoder.

```
# Transformer testing module
# Author : MOHAMED ELBASHEER
import torch
import numpy as np
import pandas as pd
import os
import math
import matplotlib.pyplot as plt
import torch.nn.functional as F
from datetime import datetime
from sklearn.metrics import precision_recall_fscore_support, confusion_matrix,
    classification_report, accuracy_score
from torch import nn, optim
from torch.utils.data import Dataset, DataLoader, TensorDataset
import torch.nn.init as init
from datetime import timedelta
import plotly.graph_objects as go
import pickle
import time
import random
import string
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
start_time = datetime.now()
# ----- 1. Need to set the model     -#

MODEL_FILE="Model_Epochs_12_Lr_1e-05_Time_2023-11-18_12-02-20.pth"
TRAINING_ERRORS="Errors_Epochs_12_Lr_1e-05_Time_2023-11-18_12-02-20.pkl"
# ----- 2. Set Testing Files  -----#

#   1) CERT r4.2:
#TEST_FILE="cert42_benign_test1_15.pt"
#TEST_FILE="cert42_benign_test2_15.pt"
#TEST_FILE="cert42_malicious_test1_28.pt"
#TEST_FILE="cert42_malicious_test2_42.pt"
```

```
#TEST_FILE="cert42_mixed_benign_15_malicious_28.pt"
#  2) CERT r5.2:
#TEST_FILE="cert52_benign_540.pt"
#TEST_FILE="cert52_benign_test1_30.pt"
#TEST_FILE="cert52_benign_test2_30.pt"
#TEST_FILE="cert52_malicious_test1_39.pt"
#TEST_FILE="cert52_malicious_test2_60.pt"
#TEST_FILE="cert52_mixed_benign_30_malicious_39.pt"
#   3) CERT r6.2:
#TEST_FILE="cert62_benign_1080.pt"
#TEST_FILE="cert62_benign_test1_60.pt"
#TEST_FILE="cert62_benign_test2_60.pt"
#TEST_FILE="cert62_malicious_test1_2.pt"
#TEST_FILE="cert62_malicious_test2_3.pt"
#TEST_FILE="cert62_mixed_benign_60_malicious_2.pt"
TEST_FILE="all_cert_versions_210_benign_and_174_malicious.pt"
# ----- 3. Set the Constants  -----#
D_MODEL = 320
NHEAD = 8
NUM_ENCODER_LAYERS = 6
DIM_FEEDFORWARD = 512
INPUT_DIM = 9 * 35  # 9 * 35 = 315
BATCH_SIZE = 1
# ----- 4. Set the Directories       -#
DATA_DIRECTORY="C:\\PRAXIS-FINAL\\TRANSFORMERS\\datasets\\"
MODEL_DIRECTORY="C:\\PRAXIS-FINAL\\TRANSFORMERS\\saved_model\\"
METRICS_DIRECTORY="C:\\PRAXIS-FINAL\\TRANSFORMERS\\metrics\\"
LOG_DIRECTORY="C:\\PRAXIS-FINAL\\TRANSFORMERS\\log\\"
TRAINING_ERRORS_DATA = MODEL_DIRECTORY  + TRAINING_ERRORS
MODEL_FILE_PATH      = MODEL_DIRECTORY  + MODEL_FILE
TEST_DATA_FILE_PATH  = DATA_DIRECTORY   + TEST_FILE
# ----- 5. Model Definition   -----#
# the model classes
class PositionalEncoding(nn.Module):
    def __init__(self, d_model, dropout=0.1, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2).float() * (-math.log
    (10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0).transpose(0, 1)
        self.register_buffer('pe', pe)
    def forward(self, x):
        x = x + self.pe[:x.size(0), :, :x.size(2)]
        return self.dropout(x)
class TransformerEncoderModel(nn.Module):
    def __init__(self, input_dim, d_model, nhead, num_encoder_layers, dim_feedforward
    , dropout=0.1):
        super(TransformerEncoderModel, self).__init__()
        self.input_projection = nn.Linear(input_dim, d_model)
        self.pos_encoder = PositionalEncoding(d_model, dropout)
        self.transformer_encoder_layer = nn.TransformerEncoderLayer(
            d_model, nhead, dim_feedforward, dropout=dropout)
        self.transformer_encoder = nn.TransformerEncoder(self.
    transformer_encoder_layer, num_encoder_layers)
        self.output_projection = nn.Linear(d_model, input_dim)  # Project back to
    original dimension
        self.layer_norm = nn.LayerNorm(d_model)
    def forward(self, src):
        src = src.view(src.size(0), src.size(1), -1).transpose(0, 1)
        src = self.input_projection(src)
        src = self.pos_encoder(src)
        src = self.transformer_encoder(src)
        src = self.layer_norm(src)
        output = self.output_projection(src)
        output = output.transpose(0, 1).view(output.size(1), output.size(0), 9, 35)
        return output
# ----- 6. Data Loader        -----#
```

```
def test_data():
    user_data_test = torch.load(TEST_DATA_FILE_PATH)
    test_data_list = [tensor for user_id, tensor in user_data_test.items()]
    test_dataset = [(tensor,) for tensor in test_data_list]
    test_loader = DataLoader(dataset=test_dataset, batch_size=BATCH_SIZE, shuffle=
    False)
    return test_loader
# ----- 8. reconstruction Functions    -#
def get_reconstruction_error_per_feature(model, dataloader):
    model.eval()
    all_errors = []
    with torch.no_grad():
        for inputs in dataloader:
            inputs = inputs[0].to(device)
            outputs = model(inputs)
            # Compute reconstruction error per feature
            error = torch.abs(outputs - inputs)
            # Sequentially apply median across the desired dimensions
            median_error = error.median(dim=1)[0].median(dim=1)[0]  # First median
    over dim=1, then over the new dim=1 (which was dim=2)
            all_errors.extend(median_error.cpu().numpy())
    return all_errors
# ----- 9. Model Loading       ----#
model = TransformerEncoderModel(9 * 35, D_MODEL, NHEAD, NUM_ENCODER_LAYERS,
    DIM_FEEDFORWARD, dropout=0.1).to(device)
model.load_state_dict(torch.load(MODEL_FILE_PATH))
model.eval()  # In evaluation mode
# ----- 10. Load the Test Data  ---#
# Load and prepare benign test data
test_loader = test_data()
# ----- 11.  Get the Reconstruction errors      #
errors_production = get_reconstruction_error_per_feature(model, test_loader)
errors_array = np.array(errors_production)
testing_errors = pd.DataFrame(errors_array, columns=[f'Feature_{i}' for i in range(
    errors_array.shape[1])])
TEST_TYPE = os.path.splitext(TEST_FILE)[0]
TESTING_ERRORS_DATA = f"{LOG_DIRECTORY}Testing_Errors_For---{TEST_FILE}---Time_{
    timestamp}.pkl"
# Save the DataFrame to a pickle file
testing_errors.to_pickle(TESTING_ERRORS_DATA)
print(f"The Testing  Reconstruction Errors.........:\n {TESTING_ERRORS_DATA}")
print(f"The Training Reconstruction Errors.........:\n {TRAINING_ERRORS_DATA}")
end_time = datetime.now()
print(f"It took {end_time - start_time} to finish running the test.")

###  <font color='cyan'> <font color='yellow'> 2.1.</font>OneClassSVM</font>
# OneClassSVM
import pandas as pd
from sklearn.svm import OneClassSVM
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings("ignore")
start_time = time.perf_counter()
# 1. load and prepare the evaluation and test data
eval_errors = pd.read_pickle(TRAINING_ERRORS_DATA)
test_errors = pd.read_pickle(TESTING_ERRORS_DATA)
# 2. scale the data (which is already scaled-just for consistency)
scaler = StandardScaler()
eval_errors_scaled = scaler.fit_transform(eval_errors)
test_errors_scaled = scaler.transform(test_errors)
# 3. now train the OneClassSVM Model
clf = OneClassSVM(gamma='auto' , nu=0.05)
clf.fit(eval_errors_scaled)
#pred_training_score=clf.score_samples(eval_errors_scaled)
# 4. Then make the predictions
predictions = clf.predict(test_errors_scaled)
anomaly_scores = clf.decision_function(test_errors_scaled)
test_errors['anomaly_score'] = anomaly_scores
# 5. Finally evaluate the results (-1 means malicious)
test_errors['prediction'] = predictions
anomalies = test_errors[test_errors['prediction'] == -1]
```

```
print(f"OneClassSVM   for {TEST_TYPE} -->  : predicted {anomalies.shape[0]} as Malicious Users and {test_errors.shape[0] - anomalies.shape[0]} as Benigns")
print(list(anomalies.prediction))
end_time = time.perf_counter()
elapsed_time = end_time - start_time
print(f"Elapsed time: {elapsed_time} seconds")
print("Parameters used in the OCSVM model:")
print(clf.get_params())
###  <font color='cyan'> <font color='yellow'> 2.2.</font>Local Outlier Factor (LOF) </font>
# Local Outlier Factor (LOF)
import pandas as pd
from sklearn.neighbors import LocalOutlierFactor
from sklearn.preprocessing import StandardScaler
start_time = time.perf_counter()
# 1. load and prepare the evaluation and test data
eval_errors = pd.read_pickle(TRAINING_ERRORS_DATA)
test_errors = pd.read_pickle(TESTING_ERRORS_DATA)
# 2. scale the data (which is already scaled-just for consistency)
scaler = StandardScaler()
eval_errors_scaled = scaler.fit_transform(eval_errors)
test_errors_scaled = scaler.transform(test_errors)
# 3. now train the LOF Model
clf = LocalOutlierFactor(n_neighbors=20, contamination=0.1, novelty=True)# original
clf = LocalOutlierFactor(n_neighbors=20, contamination=0.05, novelty=True)  # for 6.2 set 0.05  and for 5.2 set 0.01
clf.fit(eval_errors_scaled)
# 4. Then make the predictions
predictions = clf.predict(test_errors_scaled)
anomaly_scores = clf.decision_function(test_errors_scaled)
# 5. Finally evaluate the results (-1 means malicious)
test_errors['prediction'] = predictions
test_errors['anomaly_score'] = anomaly_scores
anomalies = test_errors[test_errors['prediction'] == -1]
print(f"Local Outlier Factory for {TEST_TYPE}: predicted {anomalies.shape[0]} as Malicious Users and {test_errors.shape[0] - anomalies.shape[0]} as Benigns")
print(list(anomalies.prediction))
end_time = time.perf_counter()
elapsed_time = end_time - start_time
print(f"Elapsed time: {elapsed_time} seconds")
print("Parameters used in the LOF model:")
print(clf.get_params())
###  <font color='cyan'> <font color='yellow'> 2.2.</font>Isolation Forest</font>
import pandas as pd
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
start_time = time.perf_counter()
# 1. load and prepare the evaluation and test data
eval_errors = pd.read_pickle(TRAINING_ERRORS_DATA)
test_errors = pd.read_pickle(TESTING_ERRORS_DATA)
scaler = StandardScaler()
eval_errors_scaled = scaler.fit_transform(eval_errors)
test_errors_scaled = scaler.transform(test_errors)
# 3. now train the Forest Model
clf = IsolationForest(contamination=0.1,random_state=42) #original
clf = IsolationForest(contamination=0.05,random_state=42)
clf.fit(eval_errors_scaled)
# 4. Then make the predictions
predictions = clf.predict(test_errors_scaled)
anomaly_scores = clf.decision_function(test_errors_scaled)
# 5. Finally evaluate the results (-1 means malicious)
test_errors['prediction'] = predictions
test_errors['anomaly_score'] = anomaly_scores
anomalies = test_errors[test_errors['prediction'] == -1]
print(f"Isolation Forest  for {TEST_TYPE}  : predicted {anomalies.shape[0]} as Malicious Users and {test_errors.shape[0] - anomalies.shape[0]} as Benigns")
print(list(anomalies.prediction))
end_time = time.perf_counter()
elapsed_time = end_time - start_time
print(f"Elapsed time: {elapsed_time} seconds")
print("Parameters used in the IsolationForest model:")
```

```
print(clf.get_params())
```

Source Code A.2: Transformer Testing

A.1.3 User-Based Sequencing Code

Listing A.3 provides a summary of the code used to construct and build the User-Based Sequencing Structure.

```
#!/usr/bin/env python
# Author: Mohamed Elbasheer
# This is my user based sequence for the 35 cols 4.2. The same file can be used for
# version 5.2 and 6.2. Just change the source df file
# BUILD THE USER-BASED SEQUENCING (UBS) STRUCTURE
# Build the foundation for UBS

# This is how I am constructing the User-based Sequence
# After you create the dataframe and finish all EDA Save the dataframe

UBS_DIR = "C:\\Users\\elbas\\WIP-praxis-as-of-10-25-2023\\35cols\\ubs\\"
PART_DIR = "C:\\Users\\elbas\\WIP-praxis-as-of-10-25-2023\\35cols\\part\\"
TENSOR_DIR = "C:\\Users\\elbas\\WIP-praxis-as-of-10-25-2023\\35cols\\tensor\\"
MINDEX_DIR = "C:\\Users\\elbas\\WIP-praxis-as-of-10-25-2023\\35cols\\mindex\\"
FILLER_DIR = ("C:\\Users\\elbas\\WIP-praxis-as-of-10-25-2023\\35cols\\
    mindex_with_filler_days\\")
# save the dataframe up to this point
df.to_pickle(UBS_DIR + "backup_df_just_before_scaling.pkl")
df = pd.read_pickle(UBS_DIR + "backup_df_just_before_scaling.pkl")


# Scale the Dataset
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
cols_to_scale = df.columns.difference(["user", "day", "label"])
df[cols_to_scale] = df[cols_to_scale].apply(
    lambda x: scaler.fit_transform(x.values.reshape(-1, 1)).flatten())
df.head(5)
# ------------------------------------------------------#
# --     Check for NaN values                         --#
#   scaling and some times concatenating cause NaN      #
#     when the indexes are not aligned                  #
# ------------------------------------------------------#

if not df.isna().any().any():
    print("There are no missing values in the DataFrame.")
else:
    print("There are missing values in the DataFrame.")
# ## 2.3: <font color='cyan'>Separate the labels</font>
# -- Split data based on labels (benign and malicious)   -#
benign_df = df[df["label"] == 0].drop(columns="label")
malicious_df = df[df["label"] == 1].drop(columns="label")
# ## 2.4: <font color='cyan'>Conduct simple EDA</font>
# -- Notice that the label column is no longer there     -#
print(f"Shape of the benign_df is: {benign_df.shape}")
print(f"Shape of the malicious_df is: {malicious_df.shape}")
# Optionally I can do benign_df.head() and malicious_df.head() to view the data
if not df[(df["label"] == 1) & (df.index.isin(benign_df.index))].shape[0]:
    print("There are no rows with label 1 in benign_df.index.")
# Check if malicious_df has only malicious data:
if not df[(df["label"] == 0) & (df.index.isin(malicious_df.index))].shape[0]:
    print("There are no rows with label 0 in malicious_df.index.")
# ### 2.4.1: <font color='cyan'>Save the dataframe as a checkpoint up to this point</
    font>
clean_benign_df = benign_df.copy()
clean_malicious_df = malicious_df.copy()
# write that to an pickle file
clean_benign_df.to_pickle(UBS_DIR + "backup_benign_df_ready_for_train_test_split.pkl"
    )
```

```
clean_malicious_df.to_pickle(
    UBS_DIR + "backup_malicious_df_ready_for_train_test_split.pkl"
)
# ## 2.5: <font color='yellow'>Partition the Benign and Malicious Datasets using Stratified Split</font>
# This ensures a user's sessions are entirely in the training set or entirely in the validation set, but not in both. Same things for malicious
# This is more explanation on how this means

# the groupby('user') function is applied to the DataFrame benign_df, which groups the DataFrame by the 'user' column.
# The groupby function returns pairs of (group_name, group_data), where group_name is the value of the 'user' column
# that defines the group, and group_data is the DataFrame containing all the rows of that group.
# In the list comprehension [group for _, group in benign_df.groupby('user')], the _ is used to indicate that I really
# do not want the group names (i.e., the values of the 'user' column that define the groups) in this instance; I am only
# interested in the list of DataFrames (group_data) that represent each group. simply the , _ is used as a placeholder for
# the group names, and group is used to collect the dataframes.
# The result, user_groups, is a list of DataFrames, each of which contains all the rows from df for a specific user.
# For Benign
# -----------
# first let me group by user
user_groups = [group for _, group in benign_df.groupby("user")]
# the I will slit the groups
train_groups, other_groups = train_test_split(
    user_groups, test_size=0.3, random_state=42
)
# can now simply concatenate the groups to form the datasets
benign_train = pd.concat(train_groups)
benign_other = pd.concat(other_groups)
# again group the benign_other by user and split into val and test
user_groups2 = [group for _, group in benign_other.groupby("user")]
# the I will split the groups
train_groups_val, test_groups = train_test_split(
    user_groups2, test_size=0.10, random_state=42)
# can now simply concatenate the groups to form the datasets
benign_val = pd.concat(train_groups_val)
benign_test = pd.concat(test_groups)
# now split benign_test into two groups (benign_test1 and benign_test1
user_groups3 = [group for _, group in benign_test.groupby("user")]
benign_group_test1, benign_group_test2 = train_test_split(
    user_groups3, test_size=0.5, random_state=42)
# can now simply concatenate the groups to form the datasets
benign_test1 = pd.concat(benign_group_test1)
benign_test2 = pd.concat(benign_group_test2)
# For malicious
# --------------
# first let me group by user
user_groups_mal = [group for _, group in malicious_df.groupby("user")]
# the I will slit the groups
test_mal_group1, test_mal_group2 = train_test_split(
    user_groups_mal, test_size=0.6, random_state=42)
# can now simply concatenate the groups to form the datasets
malicious_test1 = pd.concat(test_mal_group1)
malicious_test2 = pd.concat(test_mal_group2)
# ------------------------------------#
# -  Let us create the mixed group   -#
# ------------------------------------#
# concatenate benign_test1  and malicious_test1  (15 benign users and 28 malicious users)
mixed_benign_malicious = pd.concat([benign_test1, malicious_test1])
# ### 2.5.2: <font color='cyan'>Let us do basic EDA on to train/test sets</font>
print(f"benign_train shape is {benign_train.shape}")
print(f"benign_val shape is {benign_val.shape}")
print(f"benign_test1 shape is {benign_test1.shape}")
print(f"benign_test2 shape is {benign_test2.shape}")
```

```
print(f"malicious_test1 shape is {malicious_test1.shape}")
print(f"malicious_test2 shape is {malicious_test2.shape}")
print(f"mixed_benign_malicious shape is {mixed_benign_malicious.shape}")
print(f"How many benign users we have in benign_train? : {benign_train.user.nunique()}")
print(f"How many benign users we have in benign_val? : {benign_val.user.nunique()}")
print(f"How many benign users we have in benign_test1? : {benign_test1.user.nunique()}")
print(f"How many benign users we have in benign_test2? : {benign_test2.user.nunique()}")
print(
    f"How many malicious users we have in malicious_test1? : {malicious_test1.user.nunique()}"
)
print(
    f"How many malicious users we have in malicious_test2? : {malicious_test2.user.nunique()}"
)
print(
    f"How many mixed users we have in mixed_benign_malicious? : {mixed_benign_malicious.user.nunique()}"
)

# ------------------------------------#
# -  Let check if we have any NaN    -#
# ------------------------------------#
if not benign_train.isna().any().any():
    print("There are no missing values in the benign_train.")
else:
    print("There are missing values in the benign_train.")
if not benign_val.isna().any().any():
    print("There are no missing values in the benign_val.")
else:
    print("There are missing values in the benign_val.")
if not benign_test1.isna().any().any():
    print("There are no missing values in the benign_test1.")
else:
    print("There are missing values in the benign_test1.")
if not benign_test2.isna().any().any():
    print("There are no missing values in the benign_test2.")
else:
    print("There are missing values in the benign_test2.")
if not malicious_test1.isna().any().any():
    print("There are no missing values in the malicious_test1.")
else:
    print("There are missing values in the malicious_test1.")
if not malicious_test2.isna().any().any():
    print("There are no missing values in the malicious_test2.")
else:
    print("There are missing values in the malicious_test2.")
if not mixed_benign_malicious.isna().any().any():
    print("There are no missing values in the mixed_benign_malicious.")
else:
    print("There are missing values in the mixed_benign_malicious.")
# ## 2.6: <font color='cyan'>Take a backup of these train and test sets</font>
# -- Just backup those train and test sets       -#
benign_train.to_csv(PART_DIR + "benign_train.csv", index=False)
benign_val.to_csv(PART_DIR + "benign_val.csv", index=False)
benign_test1.to_csv(PART_DIR + "benign_test1.csv", index=False)
benign_test2.to_csv(PART_DIR + "benign_test2.csv", index=False)
malicious_test1.to_csv(PART_DIR + "malicious_test1.csv", index=False)
malicious_test2.to_csv(PART_DIR + "malicious_test2.csv", index=False)
mixed_benign_malicious.to_csv(PART_DIR + "mixed_benign_malicious.csv", index=False)
# ### 2.7: <font color='cyan'>Define helper functions to build Multi-Index Dataframe</font>
# -- Functions needed to build the User-Based Sequencing -#
def generate_user_day_column(data):
    data["user_day"] = data["user"].astype(str) + "_" + data["day"].astype(str)
    return data
def populate_dataframe(data, max_sessions_per_day, numeric_cols):
    # list to store the dataframe data
```

```
    all_data = []
    # I will iterate through each user and day combination
    for (user, day), group in data.groupby(["user", "day"]):
        # I will determine the number of sessions present for this user-day
    combination
        num_present_sessions = len(group)
        # I will add the rest of features to the sessions
        for idx, (_, row) in enumerate(group.iterrows()):
            all_data.append([user, day, idx] + row[numeric_cols].tolist())
        # Now: I will create a placeholders for the missing sessions. This will be
    filled with ZEROS
        #      The maximum session for each day is already chosen to be 9 (based on
    my dataset analysis)
        for session in range(num_present_sessions, max_sessions_per_day):
            placeholder_row = [user, day, session] + [0] * len(numeric_cols)
            all_data.append(placeholder_row)
    # Now I will convert the data to a dataframe with multi-index
    columns = ["user", "day", "session"] + list(numeric_cols)
    df_result = pd.DataFrame(all_data, columns=columns).set_index(
        ["user", "day", "session"])
    return df_result
# ### 2.8: <font color='cyan'>Generate 'user_day'and set naming structure</font>
# ----------------------------------------------------------#
# -- I am exclusing the "user" and "day" features         #
#        from the session dimension                        #
# ----------------------------------------------------------#
# Extract numeric columns
numeric_cols = (
    benign_df.drop(columns=["user", "day"])
    .select_dtypes(include=["float64", "int64"])
    .columns
)
# Generate 'user_day' for the entire dataset to compute the  max_sessions_per_day
df = generate_user_day_column(df)
max_sessions_per_day = df["user_day"].value_counts().max()
# Define data partitions
data_partitions = {
    "benign_train_700": benign_train,
    "benign_val_270": benign_val,
    "benign_test1_15": benign_test1,
    "benign_test2_15": benign_test2,
    "malicious_test1_28": malicious_test1,
    "malicious_test2_42": malicious_test2,
    "mixed_benign_15_malicious_28": mixed_benign_malicious,
}
# ### 2.9: <font color='cyan'>Generate the multi-index dataframe using the UBS
    functions</font>
# ----------------------------------------------------------#
# -- Build the multi-index dataframe for the user-based   #
#                    Sequencing                            #

start_time = datetime.now()
for name, df_part in data_partitions.items():
    df_part = generate_user_day_column(df_part)
    df_data = populate_dataframe(df_part, max_sessions_per_day, numeric_cols)
    df_data.to_pickle(MINDEX_DIR + f"mindex_{name}.pkl")

end_time = datetime.now()
print("\n")
print("\n")
print(
    f"It took {end_time - start_time} to finish generating the UBS multi-index
    dataframes.")
# ### 2.10: <font color='cyan'>Create a template to make all user have 500 days
    regardless if they have a session that day</font>
def process_dataframe(xyz):
    # 1. Create Template
    users = xyz.index.get_level_values("user").unique()
    days = np.arange(1, 501)
    sessions = np.arange(0, 9)
    mux = pd.MultiIndex.from_product(
```

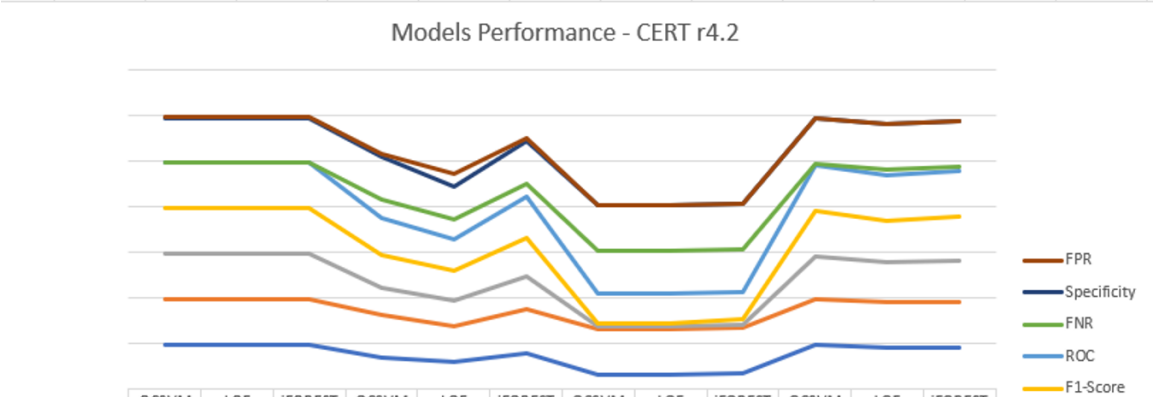
```
        [users, days, sessions], names=["user", "day", "session"]
    )
    template_df = pd.DataFrame(index=mux).reset_index()

    # 2. Merge Data
    merged_df = pd.merge(
        template_df, xyz.reset_index(), on=["user", "day", "session"], how="left"
    )
    feature_columns = [
        col for col in xyz.columns if col not in ["user", "day", "session"]
    ]
    merged_df[feature_columns] = merged_df[feature_columns].fillna(0)
    merged_df.set_index(["user", "day", "session"], inplace=True)

    return merged_df
# Read the multi-indexed pickle files generated in the previous step
read_mindex_benign_train_700 = pd.read_pickle(
    MINDEX_DIR + "mindex_benign_train_700.pkl")
read_mindex_benign_val_270 = pd.read_pickle(MINDEX_DIR + "mindex_benign_val_270.pkl")
read_mindex_benign_test1_15 = pd.read_pickle(MINDEX_DIR + "mindex_benign_test1_15.pkl
    ")
read_mindex_benign_test2_15 = pd.read_pickle(MINDEX_DIR + "mindex_benign_test2_15.pkl
    ")
read_mindex_malicious_test1_28 = pd.read_pickle(
    MINDEX_DIR + "mindex_malicious_test1_28.pkl")
read_mindex_malicious_test2_42 = pd.read_pickle(
    MINDEX_DIR + "mindex_malicious_test2_42.pkl")
read_mindex_mixed_benign_15_malicious_28 = pd.read_pickle(
    MINDEX_DIR + "mindex_mixed_benign_15_malicious_28.pkl")
# Now call the function to fill the days for every users. Simply the shape is going
    to be  500 x 9 x35
# regardless of if a user has a session on a specific day or not
filler_mindex_benign_train_700 = process_dataframe(read_mindex_benign_train_700)
filler_mindex_benign_val_270 = process_dataframe(read_mindex_benign_val_270)
filler_mindex_benign_test1_15 = process_dataframe(read_mindex_benign_test1_15)
filler_mindex_benign_test2_15 = process_dataframe(read_mindex_benign_test2_15)
filler_mindex_malicious_test1_28 = process_dataframe(read_mindex_malicious_test1_28)
filler_mindex_malicious_test2_42 = process_dataframe(read_mindex_malicious_test2_42)
filler_mindex_mixed_benign_15_malicious_28 = process_dataframe(
    read_mindex_mixed_benign_15_malicious_28)
filler_mindex_benign_train_700.to_pickle(
    FILLER_DIR + "filler_mindex_benign_train_700.pkl")
filler_mindex_benign_val_270.to_pickle(FILLER_DIR + "filler_mindex_benign_val_270.pkl
    ")
filler_mindex_benign_test1_15.to_pickle(
    FILLER_DIR + "filler_mindex_benign_test1_15.pkl")
filler_mindex_benign_test2_15.to_pickle(
    FILLER_DIR + "filler_mindex_benign_test2_15.pkl")
filler_mindex_malicious_test1_28.to_pickle(
    FILLER_DIR + "filler_mindex_malicious_test1_28.pkl")
filler_mindex_malicious_test2_42.to_pickle(
    FILLER_DIR + "filler_mindex_malicious_test2_42.pkl")
filler_mindex_mixed_benign_15_malicious_28.to_pickle(
    FILLER_DIR + "filler_mindex_mixed_benign_15_malicious_28.pkl")
# # <font color='red'>CHAPTER 3 :</font> <font color='yellow'> BUILD THE USER-BASED
    SEQUENCING TENSORS</font>
# --   I can convert the multi-index dataset         #
# --    or multi-index with the filler               #
# --   I can convert the multi-index dataset         #
# Note: Adjust the last 35  in this if you change the number of columns in a future
    dataset   ----->(-1, 9, 35)
# For now I will convert the filler multindex pickles and transforn them to tensors
# IF neeeded to convert the pickles in mindex directory, just change the  FILLER_DIR
     to MINDEX_DIR
def df_to_dict_of_tensors(df):
    tensor_dict = {}
    for user_id, user_data in df.groupby(level="user"):
        # Drop the user_id and day from the index for reshaping
        user_data_reset = user_data.reset_index(level=["user", "day"], drop=True)
        # user_tensor = torch.tensor(user_data_reset.values, dtype=torch.float32).
    view(-1, 9, len(numeric_cols))
```

```
        user_tensor = torch.tensor(user_data_reset.values, dtype=torch.float32).view(
            -1, 9, 35
        )
        tensor_dict[user_id] = user_tensor
    return tensor_dict
# ----------------------------------------#
# -- let us run and time this operation      #
# ----------------------------------------#
start_time = datetime.now()

# Get list of all .pkl files in the LOCATION_DIR
pickle_files = [f for f in os.listdir(FILLER_DIR) if f.endswith(".pkl")]
for pkl_file in pickle_files:
    # Load the multi-index DataFrame
    df_data = pd.read_pickle(os.path.join(FILLER_DIR, pkl_file))
    # Convert the DataFrame to a dictionary of tensors
    tensor_data = df_to_dict_of_tensors(df_data)
    # Save the dictionary of tensors with .pt extension
    tensor_file = pkl_file.replace(".pkl", ".pt")
    torch.save(tensor_data, os.path.join(TENSOR_DIR, tensor_file))
    print(f"Processed and saved tensor after reading the pickle file:  {pkl_file}")
    print(f"The new tensor files are saved as:  {TENSOR_DIR + tensor_file}")

end_time = datetime.now()
print("\n")
print("\n")
print(
    f"It took {end_time - start_time} to finish converting all multi-index dataframe
    to dictionary of tensors."
)
```

Source Code A.3: User-Based Sequencing Creation

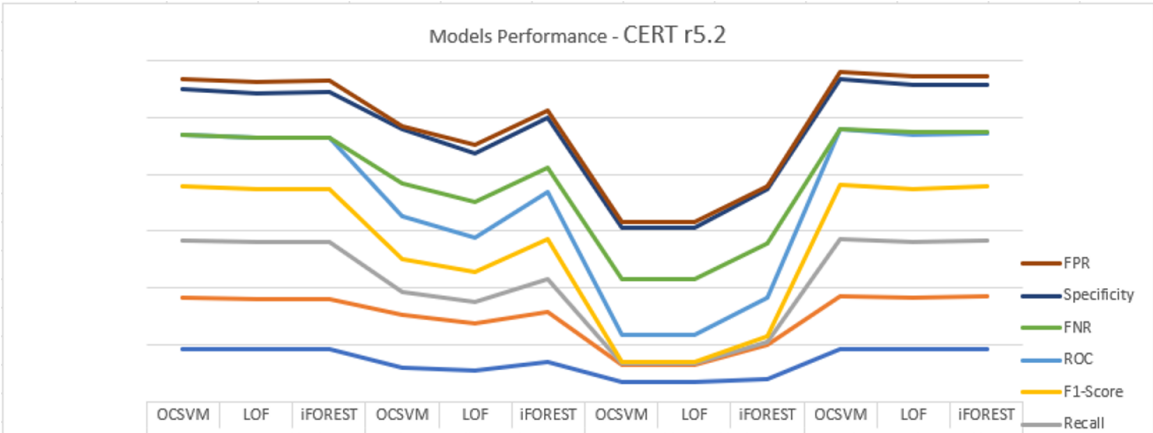
A.2 Performance Metrics

This section presents a graphical comparison of all the models used, including the standalone anomaly detection models like OCSVM, IFOREST, and LOF. The graphs provide insight into what happens if the dataset is not appropriately structured for anomaly detection, particularly in domains such as insider detection. The data clearly shows that if we take tabular data without restructuring (as in the case of the standalone model or Autoencoder-Tabular) and run it through an anomaly detection model, we cannot expect great results. However, as the data is structured correctly, we can see a significant difference in terms of better metrics, as shown in Figure A.1 for r4.2, and A.2 for r5.2, and Figure A.3 for r6.2 dataset.



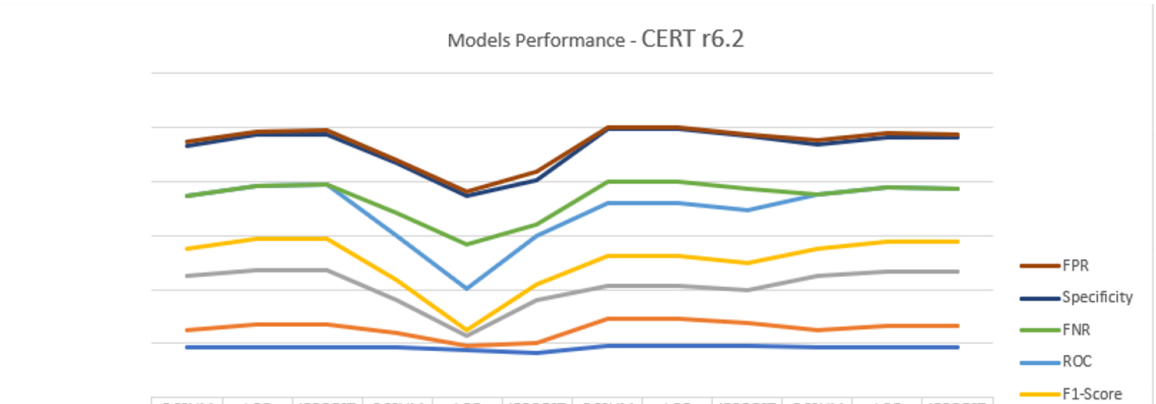
	OCSVM	LOF	iFOREST	OCSVM	LOF	iFOREST	OCSVM	LOF	iFOREST	OCSVM	LOF	iFOREST
	Transformer - UBS			Autoencoders - Tabular			Standalone -Tabular			Autoencoders -UBS		
	30 Benign and 70 Malicious (CERT 4.2)											
FPR	3%	3%	3%	7%	30%	7%	0%	0%	0%	0%	0%	0%
Specificity	97%	97%	97%	93%	70%	93%	100%	100%	100%	100%	100%	100%
FNR	0%	0%	0%	41%	46%	27%	96%	96%	93%	4%	13%	10%
ROC	100%	100%	100%	81%	69%	93%	64%	64%	58%	100%	100%	100%
F1-Score	99%	99%	99%	73%	65%	83%	8%	8%	13%	98%	93%	95%
Recall	100%	100%	100%	59%	54%	73%	4%	4%	7%	96%	87%	90%
Precision	99%	99%	99%	95%	81%	96%	100%	100%	100%	100%	100%	100%
Accuracy	99%	99%	99%	69%	59%	79%	33%	33%	35%	97%	91%	93%

Figure A.1: R4.2 performance graphs



	OCSVM	LOF	iFOREST	OCSVM	LOF	iFOREST	OCSVM	LOF	iFOREST	OCSVM	LOF	iFOREST
	Transformer - UBS			Autoencoders - Tabular			Standalone -Tabular			Autoencoders -UBS		
	60 Benign and 99 Malicious (CERT 5.2)											
FPR	18%	20%	20%	7%	15%	12%	8%	8%	5%	13%	15%	15%
Specificity	82%	80%	80%	93%	85%	88%	92%	92%	95%	87%	85%	85%
FNR	0%	0%	0%	59%	62%	42%	98%	98%	95%	1%	3%	2%
ROC	90%	88%	90%	76%	62%	84%	46%	46%	69%	98%	96%	93%
F1-Score	95%	94%	94%	57%	52%	70%	4%	4%	9%	96%	94%	95%
Recall	100%	100%	100%	41%	38%	58%	2%	2%	5%	99%	97%	98%
Precision	90%	89%	89%	91%	81%	89%	29%	29%	62%	92%	91%	92%
Accuracy	93%	92%	92%	61%	56%	69%	36%	36%	39%	94%	92%	93%

Figure A.2: R5.2 performance graphs



	OCSVM	LOF	iFOREST	OCSVM	LOF	iFOREST	OCSVM	LOF	iFOREST	OCSVM	LOF	iFOREST
	Transformer - UBS			Autoencoders - Tabular			Standalone -Tabular			Autoencoders -UBS		
	120 Benign and 5 Malicious (CERT 6.2)											
FPR	8%	6%	6%	7%	10%	17%	3%	3%	3%	8%	7%	7%
Specificity	92%	94%	94%	93%	90%	83%	97%	97%	97%	92%	93%	93%
FNR	0%	0%	0%	40%	80%	20%	40%	40%	40%	0%	0%	0%
ROC	97%	97%	98%	84%	76%	91%	98%	98%	98%	100%	100%	99%
F1-Score	50%	59%	59%	37%	11%	28%	55%	55%	50%	50%	56%	56%
Recall	100%	100%	100%	60%	20%	80%	60%	60%	60%	100%	100%	100%
Precision	33%	42%	42%	27%	8%	17%	50%	50%	43%	33%	38%	38%
Accuracy	92%	94%	94%	92%	87%	83%	96%	96%	95%	92%	94%	94%

Figure A.3: R4.2 performance graphs

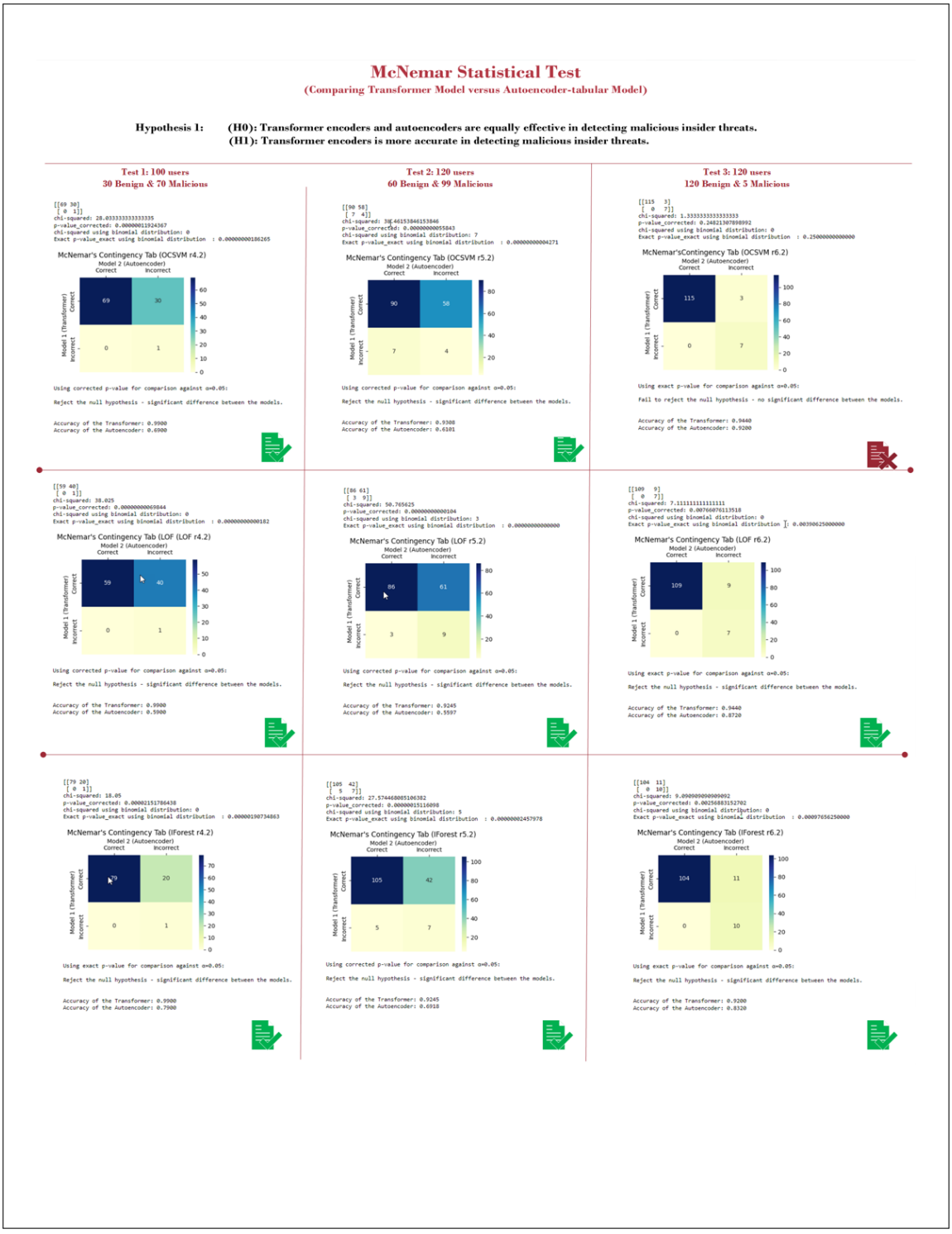
Appendix B: Hypotheses Testing

B.0.1 Transformer Versus Autoencoder-Tabular Hypothesis

Figure B.1 presents the McNemar hypothesis testing used to compare the performance of the Transformer and Autoencoder-UBS models. This is to test hypothesis 1 stating:

H1: Transformer encoders can capture long-range dependencies better than Autoencoders and accurately detect malicious insider threats.

Figure B.1: Transformer against Autoencoder-Tabular Hypothesis



B.0.2 Transformer Versus Autoencoder-UBS Hypothesis

Figure B.2 presents the McNemar hypothesis testing used to compare the performance of the Transformer and Autoencoder-UBS models. This is to test hypothesis 2 stating:

H2: User-based sequencing enables the implementation of a Transformer Encoder on tabular data to improve the detection of insider threats.

Figure B.2: Transformer against Autoencoder-UBS Hypothesis

B.0.3 Autoencoder-Tabular Versus Autoencoder-UBS Hypothesis

Figure B.3 presents the McNemar hypothesis testing used to compare the performance of the Autoencoder-tabular and Autoencoder-UBS models. This is to test hypothesis 3 stating:

H3: Using feature optimization will improve the speed and detection rate across various CERT dataset versions.

Figure B.3: Autoencoder-Tabular against Autoencoder-UBS Hypothesis