Intelligent Inside Threat Detection Framework Based on Digital Twin, Transformer Variant Models, and Transfer Learning

by

Zhi Qiang Wang

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements
for the Doctorate in Philosophy degree
in Electrical and Computer Engineering

uOttawa

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa
Ottawa, Canada

Abstract

With the rise of networked systems and modern hacker techniques, insider threats have become a greater concern than external hackers, as they often cause more significant damage and are harder to detect due to authorized access, complex behaviors, data imbalances, and a lack of explainability. To address these challenges, we proposed DTITD, a centralized learning framework that combines Digital Twin (DT) technology and transformer models. DTITD tackles data imbalance by utilizing contextual embeddings from pre-trained Large Language Models (LLMs), and it provides insights into user behavior through Digital Twin analysis, enhancing detection explainability. Extensive experiments on CERT r4.2 (dense) and CERT r6.2 (sparse) datasets show that DistilledTrans, a customized transformer model, outperforms baseline models in accuracy, precision, recall, F1-score, and AUC, while being computationally efficient. To overcome challenges like data privacy and resource costs, we introduced FedITD, a Federated Parameter-Efficient Tuning (PETuning) framework with Federated Learning (FL) and Transfer Learning. This framework allows for decentralized model learning without data transmission, safeguarding privacy and reducing resource costs. Combining DTITD and FedITD provides a highly accurate, efficient, and privacy-preserving solution for insider threat detection at an enterprise level.

Acknowledgements

I am honored to have conducted research under the guidance of Prof. Abdulmotaleb El Saddik, who welcomed me into the MCRlab. I am incredibly grateful for this opportunity. His passion, vision, and motivation have inspired me to overcome obstacles and pursue my goals. Prof. El Saddik's advice extended beyond research, providing valuable life lessons as well. I deeply appreciate his kindness, humor, and friendship. It has been a true privilege to work with him.

I would also like to thank the members of MCRlab for their support and guidance throughout my thesis work. Their feedback during lab meetings and comments to my work have always helped clarify my direction.

Finally, I am profoundly grateful to my wife and daughter for their unconditional support, both emotionally and financially. Their love, kindness, and understanding have been a constant source of inspiration, motivating me to persevere with my thesis and prepare for the future.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

BERT Bidirectional Encoder Representations from Transformers

BitFit Bias-term Fine-tuning

DistilBERT Distilled Bidirectional Encoder Representations from Transformers

DL Deep Learning

DT Digital Twin

FT Full Tuning

GPT Generative Pre-trained Transformer

LLM Large Language Model

LoRA Low-Rank Adaptation

ML Machine Learning

NLP Natural Language Processing

PETuning Parameter Efficient Tuning

RoBERTa Robustly optimized BERT approach

TL Transfer Learning

UEBA User Entity Behavior Analysis

XLNet Transformer-XL Net

Chapter 1

Introduction

1.1 Background

According to Gartner's definition, 'an **insider threat** is a malicious, careless, or negligent threat to an organization that comes from people within the organization, such as employees, former employees, contractors, or business associates, who have inside information concerning the organization's security practices, data, and computer systems'[1]

There are three common types of insider threat actors:

- Traitor — an insider that already has legal privileges but abuses their access to carry out malignant activities to the organization's resources
- Masquerader — an insider that does not have any legitimate privileges, or has lower authorization but exploits credentials to take malicious actions against the organization
- Careless user — an insider who carelessly makes mistakes to leak sensitive and/or proprietary data because of unintentional actions such as neglect or wrong configurations

In terms of malignant activities performed by the insiders, insider threat can be classified into the following three categories:

- Fraud — unauthorized inserting, deleting, or modifying an organization's data
- Data theft — reading unauthorized data or data exfiltration e.g., embezzling Intellectual Properties from the organization
- System sabotage — directly utilize information technology to sabotage or cause harm to an organization's system e.g., disruption against data integrity or service availability

In contrast, **External Intrusion** refers to unauthorized access or attempts to access a computer system, network, or data from outside of an organization's perimeter. Examples of external intrusion encompass a hacker using malware to infiltrate a company's network, phishing attacks aimed at stealing login credentials, distributed denial-of-service (DDoS) attacks to disrupt services, etc.

Insider Threats are different from External Intrusions in the following aspects:

- **Origin**: The insider threat comes from within the organization. Its sources include employees, contractors, business partners, or anyone with authorized access. However, external intrusion comes from outside the organization. Its sources encompass hackers, cybercriminals, hacktivists, nation-state actors, or other external entities.
- **Access and Trust**: Insiders have legitimate access to the organization's systems and data, making it easier for them to bypass security measures. External attackers, on the other hand, do not have authorized access and must find ways to infiltrate the organization's defenses to gain unauthorized access.
- **Motivation**: insider threats can be malicious (e.g., theft of sensitive information) or unintentional (e.g., accidental data leaks). However, external intrusion is typically malicious, including motives such as financial gain, espionage, or disruption.
- **Detection Goal**: Insider threats are often much harder to detect because the actions might appear legitimate or routine and thus are buried into a large number of

normal user behaviors. Insider threat detection's goal is to identify subtle anomalies that are nuanced deviations from normal user behavior within the context of legitimate access. However, external intrusion detection aims to find obvious anomalies that are often unusual patterns in network traffic or system behavior that suggest an external attack.

- **Detection Method**: Insider threat detection often performs internal user behavior analysis requiring considering the context of user actions, such as time of access, action temporal patterns, and relation to other users and entities. Researchers often have to track the use of privileged accounts and sensitive data access activities. In contrast, external Intrusion detection usually uses the Signature-Based Detection method, i.e., known patterns or signatures to identify external attacks or the Anomaly Detection method to monitor network traffic and system behavior for unusual activities (outliers).

- **Data Sources and Monitoring**: insider threats may include a wider range of data sources, including user activities, file usage, communication logs, and access controls, while external intrusion focuses mainly on network traffic and system-level logs, or other network-related activities. Insider threat detection analyzes user-specific data and activities within the organization, while external intrusion detection monitors network traffic and system logs for signs of external intrusion.

- **Response and Mitigation**: insider threat detection often requires detailed investigation and correlation with HR and other contextual data. It often incorporates automated responses such as account lockdown or increased monitoring to mitigate insider threats However, external intrusion detection requires immediate response to block or mitigate the threat (e.g., isolating affected systems), and continuous patching and securing of systems to prevent exploitation.

- **Prevention**: Insider threat preventive measures focus on user education, access controls, internal policies, and fostering a culture of security awareness. In contrast,

external intrusions often involve perimeter defenses, threat intelligence, and proactive security measures to timely detect and block unauthorized access attempts.

Insider threats have been a concern for organizations for decades, but their significance has grown with the increasing reliance on digital systems and data. Historically, insider threats could include physical theft or sabotage. In the digital age, however, these threats have evolved to include cyber-related activities such as data breaches, intellectual property theft, and the introduction of malicious software. The latest insider risk report [2] shows that Data theft is the primary insider threat activity. 42% of insider threat incidents are associated with intellectual property or data exfiltration. Incidental or unintentional disclosure accounts for twenty-three percent. 19% were caused by system sabotage, and nine percent were due to fraud. This thesis focuses on insider threat detection.

The increasing complexity and interconnection of IT systems, combined with the growing volume of sensitive data stored and processed by organizations, have made insider threats a more critical issue. Several factors contribute to the seriousness of these threats:

- **Frequency**: Insider threats are increasingly common. According to the 2022 Cost of Insider Threats Global Report by the Ponemon Institute [3], the frequency of incidents involving insider threats has risen significantly. The report indicates that the number of incidents has increased by 47% over the past two years.

- **Severity**: The impact of insider threats is often more severe compared to external attacks. Insiders have intimate knowledge of the organization's systems and can exploit this knowledge to cause significant harm. The same report by the Ponemon Institute found that the average cost of an insider threat incident is $15.38 million, which is considerably higher than many external breaches.

- **Duration**: Insider threats tend to be prolonged, with many incidents taking months or even years to detect. This extended duration allows for more substantial

damage to be inflicted over time. On average, it takes organizations 85 days to contain an insider incident, compared to 23 days for external threats.

- **Range of Impact**: Insider threats can affect various aspects of an organization, including financial loss, operational disruption, reputational damage, and legal consequences. For instance, a single insider incident can result in the theft of sensitive intellectual property, which could undermine a company's competitive advantage or lead to substantial legal penalties and regulatory fines.

Organizations now recognize that internal actors can often bypass many of the external security measures, making the detection and prevention of insider threats a crucial aspect of overall security strategy. According to Insider Threat Report 2019, about 60% organizations experienced one or more insider attacks in 2019 [4]. Because an insider has permitted access to an organization's system, it is possible to cause larger losses and greater damage than an external hacker. The 2018 U.S. State of Cybercrime Survey indicates that 30% of respondents showed incidents produced by insider attacks are more expensive or harmful than external attacks [5]. Organizations spent an average of $11.45 million in 2020 to handle insider threats while they paid $15.38 million in 2021 that rose 34% in one year [6].

1.2 Motivation

Researching insider threat detection is of great significance today:

- **Protection of Sensitive Information**: Insider threats often involve individuals with authorized access to sensitive data. Detecting these threats is crucial to prevent unauthorized disclosure of proprietary information, trade secrets, personal data, and classified government information.

- **Financial Security**: Insider threats can lead to significant financial losses through fraud, theft, and sabotage. Detecting such threats helps organizations safeguard

their financial assets and avoid costly breaches.

- **Operational Continuity**: Insiders with malicious intent can disrupt business operations by damaging critical systems, deleting essential data, or obstructing workflows. Effective detection helps maintain operational stability and business continuity.

- **Regulatory Compliance**: Many industries are subject to strict regulatory requirements regarding data protection and privacy. Insider threat detection helps organizations comply with these regulations and avoid legal penalties.

- **Reputation Management**: Data breaches and security incidents caused by insiders can severely damage an organization's reputation. Proactive detection of insider threats helps protect the organization's image and maintain trust with customers, partners, and stakeholders.

- **Employee Safety**: Insiders can also pose physical threats to other employees through acts of violence or sabotage. Detecting such threats ensures a safe working environment.

Therefore, we have strong motivations to delve into the research of detecting internal threats:

1. **Improvement of Detection Technologies**: Current detection systems often generate false positives and may miss subtle indicators of insider threats. Research is driven by the need to enhance the accuracy, efficiency, and reliability of detection technologies. As organizations adopt more complex and interconnected technologies, the attack surface for insider threats expands. Research in this field is also motivated by the need to develop advanced detection techniques that can keep pace with evolving threats.

2. **Privacy Concerns**: Insider threat detection must balance security with respect

for employee privacy. Research is motivated by the challenge of developing methods that are effective yet minimally invasive and preserve privacy.

3. **Proactive Defense**: The ultimate goal of insider threat detection research is to shift from reactive to proactive defense strategies, identifying and mitigating threats before they can cause harm. This proactive approach is critical to minimize damage and improve overall security posture.

4. **Cost-Effectiveness**: Developing cost-effective detection methods is a major motivation, as many organizations, especially smaller ones, may not have the resources to implement expensive security solutions. The research aims to create accessible and scalable detection systems.

5. **Behavioral Understanding**: Understanding the behaviors and motivations of insiders is crucial for developing effective detection mechanisms. The research aims to identify behavioral patterns and indicators that precede malicious actions

6. **Integration of Multidisciplinary Approaches**: Insider threat detection benefits from a combination of disciplines, including cybersecurity, psychology, data science, AI, and organizational behavior. Research motivated by this interdisciplinary approach seeks to create more holistic and effective solutions.

In summary, the importance of insider threat detection lies in its ability to protect assets of an organization, ensure compliance, and maintain operational integrity, while the motivation for ongoing research is driven by the need to keep up with evolving threats, improve detection accuracy, and balance security with privacy considerations.

1.3 Existing Problems

Preventing insider threats requires a multifaceted approach that combines technical, procedural, and cultural measures. Key strategies include 1) Implementing strict access controls, such as the Principle of Least Privilege and Role-Based Access Control; 2)

Monitoring user activities with software tools to detect anomaly behavior; 3) Security awareness training and clear communication of internal policies help educate employees;4)Background checks and Employee Assistance Programs (EAPs) address psychological and behavioral risks; 5) A well-defined incident response plan; 6) Regular audits of access controls and security measures are essential for early detection and response to potential threats. This thesis focuses on insider threat detection. However, detecting insider threats faces the following significant problems and challenges:

1. **High Data Scarcity**: Due to isolated data islands and concerns about security and privacy, organizations are more reluctant to share insider threat data, resulting in a lack of anomaly data to effectively train robust detection models. Additionally, the data are often homogeneous with little variance and representative. This leads to a biased model. Moreover, it is difficult for them to detect new or unknown threats because these threats never happen locally.

2. **Highly Imbalanced Data**: Clients often have a large number of records of normal user behaviors but few available anomalous data instances. Therefore, they usually lack enough anomaly data to train robust insider threat detection models effectively. This leads to biased models with notably lower sensitivity and a high false positive rate. Additionally, the collection and update of anomaly data are slow and difficult because of legal and regulation reasons.

3. **Poor Detection Performance**: Compared with external intrusions, insider threats are often much harder to detect. Insiders often have legitimate privileges to access an organization's system, making it easier for them to bypass security measures. Thus malicious activities disguise themselves within a multitude of normal user behaviors and might appear legitimate or routine. Insider threat models have to identify subtle anomalies that are small deviations from normal user behavior rather than obvious anomalies which are often unusual patterns in network traffic or system behavior suggesting an external attack. Additionally, user behavior

patterns and correlations are so complex, deep, and hidden that it requires more powerful models to have a deeper understanding of the context of user activities. Furthermore, most models are supervised learning models. Training such models requires a great amount of labeled data, which can be scarce and difficult to obtain in the real world.

4. **High Resource Cost**: Insiders' behavior has obvious temporal patterns but the patterns may drift or shift over time. However, most existing modeling methods focus on behavior category features such as various frequencies or social networks but ignore temporal patterns. Although some models attempt to capture temporal relationships, they are unable to handle long temporal sequences. The remaining models often incorporate multiple deep learning models, leading to excessively complex, resource-intensive, and time-consuming training. They bring about high communication costs, memory usage, and storage room. The resource constraints limit their applications in edge computing and collaborative learning.

5. **High Data Heterogeneity**: The traditional insider threat detection systems lack the capability of personalization. Most methods rely on a global model for nearly all clients. Once enough client data is gathered to train a satisfactory global model, it is for all clients to detect intrusion. However, various organizations may have different user daily behavior patterns and face different security threats. Therefore, a global model is not practical and custom models should be considered for insider threats. Additionally, clients often lack labeled data.

6. **Long Latency and High Bandwidth Usage**: traditional insider threat detection methods often involve aggregating data in a central location for analysis, leading to long latency and high bandwidth usage. However, customers increasingly demand local real-time detection systems with locally trained models and prompt alert generation for insider threats.

7. **Data Safety and Privacy**: centrally aggregating data in a central location for

model training and inferencing a global model for all clients also creates a single point of failure and a larger risk surface for all clients. Clients concerned about their data privacy and security

8. **Lack Explanation**: Traditional insider threat detection methods lack interpretability. Insider threat detection often requires a deeper understanding of internal users' and entities' behaviors and related correlations. It requires considering the context of user actions, such as time of access, action types, and relation to other users and entities. However, we lack the tools to illustrate them to facilitate in-depth analysis. it is also not easy to explain the detection results and decisions, for example, why get the results and what needs to be changed to mitigate the risks.

1.4 Contributions

We aim to improve insider threat detection using two main approaches: the central learning solution by proposing a framework based on Digital Twins (DT) and full-tuning various transformer variant models and the federated learning solution by proposing a framework based on Parameter Efficient Tuning (PETuning), pre-trained Large Language Models(LLMs) and transfer learning unsupervised domain adaption method.

1.4.1 Central Learning Framework - DTITD

To overcome the above challenges 2, 3, and 8, this thesis proposes DTITS, a novel intelligent framework for Insider Threat Detection based on Digital Twins (DT) and Self-attention Based Deep Learning Models. To overcome the highly imbalanced data issue, this thesis applys Natural Language Processing (NLP) data augmentation methods to upsample minorities. To improve poor detection performance and accurately identify anomalies, this study adopts transformer variant models (Original Transformer, DistilledTrans, BERT + Final Layer, RoBERTa + Final Layer, BERT + CNN, BERT + LSTM). To increase explanation, Digital Twins are applied to provide transparency

and interpretability to insider threat detection.

The DTITS comprises three parts:

1. **DATA PREPARATION SECTION**: This section is composed of an ETL pipeline, Data Stage, and Data preprocessing module. The ETL pipeline module collects and forwards the related log or contextual data to the Data Stage. The data preprocessing module cleanses data, extracts features, designates a numeric token to user activity type and occurrence time, and builds the daily behavior vector for each user. It can also perform contextual word embedding insert and substitution, or context sentence embedding using a pre-trained model, such as Bidirectional Encoder Representations from Transformers (BERT) or Generative Pre-trained Transformer 2 (GPT 2), to augment the minority class in imbalanced datasets.

2. **MODEL CONSTRUCTION SECTION**: This section consists of a DT service, scenario definition, custom model building (mode training, validation, and test) module, and the fine-tuning module of the pre-trained models. This paper proposed a customized transformer named DistilledTrans and applied the original transformer model, pre-trained transformer model (BERT, RoBERTa) plus a final layer, and the hybrid methods combining a pre-trained transformer model (BERT, RoBERTa) with Convolutional Neural Network(CNN) or Long Short Term Memory(LSTM) models to detect insider threats. **Hybrid model** in the context of this study refers to a combination of different deep learning models, especially a combination of transformer model and other deep learning models.

3. **INSIDER THREAT DETECTION SECTION**: This section is composed of a model inference pipeline, dashboard, and remediation module. The model inference module utilizes a well-tuned trained model to detect whether an insider's activity is abnormal. If yes, the remediation module uses predefined strategies to take measures to disrupt or deter the user's activity to remediate the insider risk and

notify the related stakeholders.

Contributions: the main contributions of the DTITD framework are as follows:

- **Innovative Insider Threat Detection Framework:** To our knowledge, this work is one of the first attempts to learn the behavior patterns of insider users using the Digital Twin and Natural Language Processing (NLP) deep learning method based on the self-attention mechanism. This is an innovative all-in-one end-to-end insider threat detection solution based on AI.

- **Effective Data Augmentation:** This study effectively overcomes the issue of highly imbalanced data by performing novel language data augmentation approaches: contextual word embedding insert and substitution predicted by the BERT model, and contextual embedding sentence predicted by the GPT-2 model. This study proves that the latter is a more effective data augmentation approach for detecting insider threats.

- **Simplified Transformer Model DistilledTrans:** This thesis proposes a simplified encoder of the transformer model called DistilledTrans. The experimental results reveal that DistilledTrans trained with sparse dataset CERT r6.2 augmented by contextual embedding sentences outperforms the state-of-the-art models in terms of all evaluation metrics, including accuracy, precision, recall, F1-score, and Area Under the ROC Curve (AUC). Additionally, its structure is much simpler, and thus the training time and computing cost are much less than the recent models.

- **Pre-trained Models for Dense Data:** When training models with the dense dataset CERT r4.2, pre-trained models BERT plus a final layer or RoBERTa plus a final layer can achieve significantly higher performance than all the current models with only a very little sacrifice of precision. Meanwhile, the models are much more concise and simpler than all existing deep learning models. This proves that

the self-attention structure is a good substitute for convolutional and recursive mechanisms rather than a supplement. The hybrid models are not necessary for insider threat detection.

- **Digital Twin:** This framework introduces DT service. It can not only timely and continuously monitor the insider risk profile of an organization but also easily perform in-depth analysis to find root causes, clearly explain the results, and quickly take appropriate remediation actions against insider threats.

1.4.2 Federated Learning Framework - FedITD

To address the existing problems 1, 3, 4, 5, 6, and 7, this thesis proposes FedITD, an innovative intelligent framework for Insider Threat Detection based on Federated Parameter Efficient Tuning with Pre-trained Language Model and Transfer Learning. To improve poor detection performance and accurately identify anomalies, this thesis combines pre-trained LLMs (BERT, RoBERTa, XLNet, DistilBERT), PETuning, and transfer learning. To break the barriers between isolated data islands and solve data scarcity, this study utilizes federated learning, PETuning, and transfer learning to indirectly learn from multiple clients' data while ensuring data privacy and security. To handle data heterogeneity and the lack of labeled data, the thesis incorporates a transfer learning unsupervised domain adaptation method instead of collecting more labeled local data and retraining a new model. To reduce high resource costs, PETuning methods are applied (Adapter, BitFit, LoRA) instead of full tuning in federated learning. To address long latency and high bandwidth issues, this thesis proposes and implements a flexible, hierarchical, and federated framework named FedITD, with local real-time detection systems, locally trained models, and federated PETuning methods.

The FedITD is composed of four components:

1. **Federated Learning**: To overcome this obstacle of data integration, federated learning (FL) [7] has surfaced as a privacy-conscious technique. FL aims to col-

lectively train models without the need to transmit substantial client data to a centralized location. In FL, decentralized clients only simply compute and send a small part of model information, like parameters or gradients, to a server at intervals, where they are then aggregated to generate a global model. It disrupts data islands by enabling learning across distributed data sources. It accelerates training and minimizes communication requirements compared to centralized learning methods. Additionally, it facilitates collaboration without compromising the security of each participant by maintaining data locally. Thus, FL is chosen as our system's foundation.

2. **Pre-trained Language Models**: Pre-trained LLMs, like BERT [8] and RoBERTa [9], XLNet [10], and DistllBert [11] have showcased remarkable performance across various natural language processing (NLP) tasks, such as GLUE etc. Wang et al.[12] demonstrated Pre-trained LLMs' exceptional performance of insider threat detection by centrally full-tuning the Pre-trained LLMs on the CERT dataset r6.2 and CERT dataset r4.2. Therefore, we adopt Pre-trained LLMs as the backbone models of our FL system. However, large LLMs introduce challenges such as significant communication overhead, storage costs, and high costs associated with adapting local models after federated Learning and security concerns.

3. **Parameter Efficient Tuning**: Parameter Efficient Tuning (PETuning) methods, including Adapter tuning [13], LoRA[14], and BitFit [15], typically involve locking most of parameters in Pre-trained LLMs and only tuning a small subset of extra parameters or a small portion of the initial model parameters for downstream jobs. These methods were originally trained centrally. This study introduces them to federated learning to effectively reduce communication costs and resource usage while maintaining satisfactory performance. It also can significantly improve security e.g., counter gradient inversion attacks, which can result in an average reduction of 40.7% in the accuracy of recovered words compared to Federated Parameter Full

Tuning [16].

4. **Transfer Learning**: However, the global model may not fit a specific client well due to local data heterogeneity and diversity. Using Transfer Learning (TL), especially unsupervised domain adaption, this study can rapidly develop a custom local model tailored to a specific organization or case, leveraging insights gained from previous experiences shared by other organizations. The central concept is to minimize the distribution divergence between various domains. It significantly enhances adaptability because this approach could offer multiple variations of the same global models customized for different organizations or use cases with their limited and unlabeled local data, thus yielding superior local results. We combine federated PETuing and TL to address the aforementioned challenges and further improve unknown attack detection.

Contributions: this study contributes in the following ways:

- **Novel Framework**: To the best of our knowledge, FedITD is the first framework to leverage the FL of transformer variant models, PETuning, and TF for insider threat detection. This paper proposed a flexible, hierarchical, and federated framework with local real-time detection systems, locally trained models, and federated PETuning methods. This framework indirectly learns from multiple distributed clients' data without compromising privacy. It not only successfully overcome high data scarcity and data homogeneousness issue but also effectively leading to low latency and prompt alerts. The study also explored alternative PETuning implementations like Adapter and LoRA.

- **Excellent Detection Performance**: FedITD outperforms other federated learning methods and almost all central training-based insider threat detection methods. It closely matches the best centrally trained method, DistilledTrans. Additionally, it demonstrated FedITD's strong adaptability to both highly and slightly heterogeneous data and effectiveness in detecting new or unknown attacks by indirect

learning from other clients without compromising privacy.

- **High Resource Efficiency**: FedITD Demonstrated significant reductions in communication, memory, and storage costs using Federated PETuning compared to full-tuned original LLMs.

- **Optimal Equilibrium**: this study explored the system configuration to attain the optimal TL performance across source data domain, unlabeled local data domain, and global data, which other methods lack.

1.5 Scholarly Achievements

- **Refereed book and journal papers:**

  1. **Z. Wang**, 2022. "Architecture Reference Models of Digital Twins for Healthcare". In A. E. Saddik, "Digital Twin for Healthcare: Design, Challenges, and Solutions," Elsevier, 2022.
  2. **Z. Wang** and A. E. Saddik, "DTITD: An Intelligent Insider Threat Detection Framework Based on Digital Twin and Self-Attention Based Deep Learning Models," IEEE Access, 2023 Oct 13.
  3. **Z. Wang**, H.Wang, and A. E. Saddik, "FedITD: A Federated Parameter-Efficient Tuning with Pre-trained Language Models and Transfer Learning Framework for Insider Threat Detection," IEEE Access, vol. 12, pp. 160396-160417, 2024, doi: 10.1109/ACCESS.2024.3482988.

1.6 Thesis Organization

The thesis is organized as follows:

- Chapter 2 introduces the related works of insider threat detection, which mainly contains Machine Learning methods, Deep Learning approaches, Digital Twin,

Transformer models, hybrid models with transformer models with other deep learning models, federated learning, Parameter Efficient Tuning, and unsupervised domain adaption of transfer learning.

- Chapter 3 presents the proposed central learning framework: an Intelligent Insider Threat Detection Framework Based on Digital Twin and Self-attention Based Deep Learning Models
- Chapter 4 presents the proposed federated learning framework: a Federated Parameter-Efficient Tuning with Pre-trained Language Models and Transfer Learning Framework for Insider Threat Detection
- Chapter 5 summarizes the thesis content and contributions, and briefly discusses possible future research directions.

Chapter 2

Background and Related Work

In this chapter, we provide an overview of the related work to insider threat detection including rule based methods, machine learning based methods, deep learning based methods, federated learning, Parameter Efficient Tuning, and transfer learning. We first review the rule based methods for insider threat detection in section 2.1, where most work mainly focuses on signatures. Then, we discuss the machine learning based methods on host data, network data, and contextual data in section 2.2. The related works about deep learning algorithms proposed for insider risks and their pros and cons are reviewed in section 2.3. Next, the overview of Federated Learning is presented in section 2.4 and Parameter Efficient Tuning is introduced in section 2.5. Finally, transfer learning especially unsupervised domain adaption is shown in section 2.6.

2.1 Existing Insider Threat Detection Methods

2.1.1 Rule Based Methods

Initially, researchers used a rule-based method to detect insider threats. Nguyen et al. [17] applied rules to detect data exfiltration using signature matching. For instance, if a user accesses more frequently than the threshold or access from an illegal location, the user and his/her activity will be labeled as an anomaly. Hanley and Montelibano [18]

proposed a set of hands-on rules for identifying data exfiltration based on Email, Human Resource(HR), and Lightweight Directory Access Protocol(LDAP) data. ELICIT [19] was introduced to utilize hand-coded rules and signature matching to detect insiders based on contextual data and network traffic. However, rule-based methods require deep domain expertise to perform in-depth analytics to identify signatures of insider threats. The response to insider risk is relatively slow because it is an after-damage solution. With the rapid growth of unknown insider threats, even if the known threat had little variation, it could not be detected because its signature was not known.

2.1.2 Machine Learning Methods

In the last decade, lots of researchers tried to apply a machine learning model to detect insider threats. This thesis summarized the machine learning models' related works according to their data sources: host data, network data, and context data.

- **Host data** are gathered from each computer or server including system calls, commands, and host logs. Maxion et al. [20] applied Naïve classification to designate posterior probabilities to every test command based on users' historical data correspondingly. Next, they judged if a command sequence came from this user by checking the cumulative posterior probabilities. Salem et al. [21], [22] proposed two systems quantifying the abnormality with the Hellinger Distance. Every subsequent command was converted into a binary vector in agreement with whether a command happened historically or not. Then a one-class SVM was trained to detect the anormal subsequence according to the frequency of command. Kudłacik et al. [23] presented a solution for building a fuzzy user profile on the frequency vector of a user's command. All local user profiles could accurately compose this user's daily behavior and temporal pattern. The average probability of a test command could determine if a command sequence is abnormal. Song et al. [24] introduced a behavioral biometrics-motivated system to prevent insider threats. They extracted 18 features from gathered logs for profiling the pattern of a user's behavior e.g., the

number of files accessed. They used the Gaussian mixture model (GMM), SVM, and kernel density estimation (KDE) to detect. Lastly, they found that the GMM performs the best. However, GMMs are limited for insider threat detection due to their inability to capture temporal sequences, contextual changes, and user behavior evolution. They treat data points independently, leading to missed patterns and high false positives in complex, overlapping behaviors. GMMs also struggle with categorical data, common in insider threat scenarios.

- **Network data** are collected from network logs including Proxy, firewall, VPN, LDAP, HTTP browser, and Email. DISCLOURE is proposed by Bilge et al. [25] to detect botnets by analyzing network flow as a whole. Extracted features include flow attribution, client access habits, and temporal information. They performed Classification Algorithms such as the decision tree, SVM, Random Forest model, and an ensemble of multiple sub-detectors to detect malicious activities. In [26] and [27], one detector was built on a specific type of network log and then they were combined into an ensemble model to detect threats so that the information across different types of network login the system was allied. More than 100 features were extracted from a great variety of network logs including Email, proxy, LDAP, etc. Several machine learning detectors including Markov model, Logistic Regression (LR), KNN, etc. deal with a particular subgroup of the features. Lastly, they allocated a score to this user and entity in terms of the number of malicious activities. An alternative method of handling network logs is to employ a machine learning system to handle a full set of features obtained from various types of network logs. The Beehive [28] applied Principal Component Analysis(PCA) to reduce the dimensions of features and then chose 15 features on a per day per host basis. Next, they applied the k-means clustering model to detect abnormal behaviors. A graph-based system was also proposed to address insider threats. [29] They used a bipartite graph approach to model users' relationship with data from various network logs, file systems, and psychological surveys. Then they utilized

Isolation Forest (IF) to detect malicious users from graphs. Le et al. [30] employed machine learning algorithms including Logistic Regression, Neural Network, Random Forest, and XGBoost to detect insider threats. Random Forest performs best at a detection rate of 62%, precision of 5.96%, and F1 score of 10.10% on the CERT r6.2 dataset. They also could not detect scenario 5. Al-Shehari et al. [31] applied various random sampling methods, including under-sampling, over-sampling, and hybrid sampling, to deal with the highly imbalanced classes of the CERT r4.2 dataset. They then trained several machine learning models, including Extreme Gradient Boosting (XGB), Decision Tree (DT), Random Forest (RF), and K-Nearest Neighbor (KNN), on these balanced datasets to detect insider data leakage. The results were compared with those of the baselines (the same models trained by the imbalanced datasets) and the existing models. The experimental results demonstrated that their models improved the detection performance by effectively solving the class imbalance problem.

- **Contextual data** is the contextual information of a human user e.g., HR and psychological data. Brdiczka et al. [32] applied Structural Anomaly Detection (SAD) to detect anomalies from social networks. The sequential Bayesian model was used to model users' connections. Then, they improved users' psychological profiles by using three features (motivation, personality, and emotional state). Finally, they combined users' connections and psychological features to generate an abnormal score for the user.

However, these machine learning (ML) based model structures are too shallow to capture complicated user behavior patterns that are often nonlinear and hidden. This often leads to a high negative positive rate and a low detection rate. In addition, they require handmade feature engineering, which is time-consuming and requires domain expertise. Therefore, an increasing number of scholars have turned to deep learning models to handle insider threats. Compared to ML models, deep learning models can capture deep and salient nonlinear correlations to learn complex user behavior patterns. For

Table 2.1: The Machine Learning Methods of Insider Threat Detection

Data Source	**Data Type**	**Model**	**Reference**
Host	Command Sequence	Naïve Bayes Classifier	[20]
		SVM	[21] [22]
		Fuzzy Logic	[23]
	Biometrics	Bipartite Graph	[24]
Network	Network Traffic	Ensemble of Decision Tree, SVM and Random Forest	[25]
	Network Log	Ensemble of Markov Model, LR, and KNN	[26] [27]
		K-Means	[28]
		Bipartite Graph	[29]
		XGB, Decition Tree, Random Forest, KNN	[31]
Context	HR and Psychological Data	Sequential Bayesian	[32]

example, some deep learning models have demonstrated good performance on sequential data, such as user activity sequences. Thus, it is a good fit for modeling complex user behaviors and accurately identifying abnormal users. In addition, deep learning models make it easier to integrate heterogeneous data sources across multiple domains. Moreover, feature engineering is relatively simpler, without too much prior knowledge and human labor efforts. Furthermore, some unsupervised learning approaches have strong learning capabilities without labeling works.

2.1.3 Deep Learning Methods

In this thesis, we generally classify deep learning model-related works according to their model architectures: deep feedforward neural network (FNN), convolutional neural network(CNN), recurrent neural network(RNN), graphic convolutional network (GCN), and transformers.

FNN: Lin et al. [33] proposed an unsupervised hybrid Deep Belief Network (DBN) based model to detect insider threat. The model is built by multilayer Restricted Boltzmann Machines (RBMs) piled together. Extract features from audit logs in five domains (logon, device, email, http, and file log data) and load them into the model. Then, the output of learned features from the last salient layer is fed into the One-Class Support Vector Machine (OCSVM) model for insider threat detection. It is a good fit for the solution of combining multidomain features rather than a single domain to analyze user behavior patterns. However, it cannot model temporal data and produces a lot of false alarms due to information loss in feature extraction. An ensemble of multiple deep autoencoder-based models was proposed in Lin et al. [34]. The training data are from logon, file, device, and http logs. Features from each data source are fed to the respective deep Autoencoders for modeling the normal behavior of users. The deep autoencoders intend to minimize the reconstruction error. The outputs that have relatively higher reconstruction errors than the threshold between input data and decoded data are assembled to compose an ensemble. A user's overall maliciousness score is jointly

calculated based on them.

This FNN approach effectively enhanced the accuracy with a low negative positive rate without prior expertise because the cascaded multiple encoder and decoder can capture the hidden nonlinear relationship between user behaviors. However, since most features are simple aggregated counts of activities, which are frequency-based, the approach is too general for insider threat detection because it still cannot capture temporal patterns. Additionally, the method often uses data late fusion building models on each category of data first and then incorporating the outputs rather than fusing data at the beginning. Therefore, the approach loses the opportunity to capture the correlation across multiple domains in the early phase.

RNN: User activities ordered by time within a time window can be generally regarded as a time sequence. To capture temporal patterns, RNN-based models have attracted increasing interest. The basic concept is that an RNN model is trained to predict the subsequent activities of a user. If the prediction results deviate substantially from the user's real activities, the user's activities may be anomalous. However, the classic RNN is not easy to train, especially for long sequences, because of the exploding or vanishing gradient. Researchers often use Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) instead to model temporal patterns. An unsupervised insider threat detection system was proposed by Tutor et al. [35]: the Feature Extraction module aggregates the counts of user activity from various sources (logon, device, email, http, and file log data) and outputs a temporal vector per user per day. The LSTM model and density clustering model took it as input and output a series of hidden state vectors. Then, the outputs were input into the structured prediction model to compute the anomaly scores using the negative log probability of user actions. This type of model is a more appropriate method for Insider Threat Detection, as it can effectively learn temporal user behavior patterns by using hidden units from normal temporal sequences and identify malicious behavior when the activity is essentially different from normal behavior predicted from the trained model. Additionally, the model can be gradually updated online;

thus, it can automatically evolve with new patterns over time. However, this method is too slow to achieve long-term temporal dependency. Moreover, it can find outliers in users' behaviors but cannot assure whether they are true positives with high confidence. Furthermore, it may fail to detect malicious activities within one day. In contrast to this method, the system proposed by Lu and Wong [36] not only has a historical user behavior analytics component similar to this method but also has an online monitor and real-time detection component based on LSTM. The model can use a time sequence as an input to detect insider threats and self-evolve during the online monitoring process. More context data were extracted from the CERT dataset to enrich the feature set, such as employees' roles, positions, personalities, and relationships with other employees. All of them increase confidence in determining insider threats. Yuan et al. [37] trained an LSTM model to predict the next user's activity and obtained a group of temporal hidden states to produce a fixed-size feature matrix, which is fed into the CNN classifier to generate the probability of abnormal behavior based on the discrepancy from normal activity sequences. Hence, their model can detect abnormal behaviors occurring in a single day. Yuan et al. [38] designed a hierarchical neural temporal point-process model to detect insider threats. Their system captured the users' behaviors, including not only activity types but also time information at two levels: a lower-level LSTM was used to predict whether the next session was malicious by learning intra-session information, for example, activity types and activity time intervals on the intra-session level, while an upper-level LSTM was used to learn inter-session information to predict the intervals between two sessions and the duration of the next session. Finally, they computed an anomalous score according to the gap between the predicted results and the real action time and type. Haq et al. [39] adopted the word embedding methods Word2vec and GLoVe to encode user activity information and then fed it into the LSTM model. Next, they compared state-of-the-art ML models such as eXtreme Gradient Boosting (XGBoost), Ada-Boost, Random Forest (RF), K-nearest Neighbors algorithm (KNN), and Logistic Regression(LR). Their results showed that XGBoost performed best in terms

of accuracy (92%). Nevertheless, their word2vec+ LSTM and GLoVe + LSTM models achieved accuracies of only 73.4% and 74.0%, respectively. Singh et al. [40] introduced a hybrid learning method for detecting insider threats. They utilized a bidirectional LSTM model to extract features, a feed-forward artificial neural network to select relevant features using distance measurements, Euclidean distance, Cosine similarity, and standard deviation, and an SVM classifier to detect abnormal users. Meanwhile, they applied the genetic algorithm's fast global search strategy for the SVM's initial kernel selection to enhance its performance. Finally, alerts were triggered for every user if their aggregated anomaly scores exceeded the threshold.

Overall, RNN is still likely to fail to learn long-term temporal patterns. In addition, training an RNN is time-consuming because the RNN structure is too complicated and it cannot make use of the parallel computing capacity of the GPU but takes words in sequence.

CNN: CNN model performs well in computer vision because whether an image or voice is Euclidean data with a regular spatial structure. However, it is not suitable for capturing temporal sequence patterns, particularly time series. In addition, it requires a fixed-size feature matrix as input. Saaudi et al.[41] proposed a solution consisting of CNN and LSTM. The CNN was composed of two layers: a convolutional layer with ReLU and a max-pooling layer. Each log sequence (logon, file, http, device, email logs) was combined and then transformed from a list of strings to a list of the character indexes with the same length through embedding and padded by 0. These feature map matrices were then fed into the CNN. Next, they applied LSTM to model the temporal patterns between the extracted features. Finally, a regular dense neural network layer was utilized to classify the users as normal or malicious. The experimental results showed that the CNN-LSTM model outperformed the CNN and LSTM models. This approach takes advantage of the merits of CNN for capturing the local context and LSTM for modeling the temporal pattern. It does not require handcrafted features and implements automatic and fast feature representation learning.

However, the CNN-based model still easily loses some valuable information because of the pooling layer and requires a large dataset with labels for training. In addition, it is not suitable for modeling features without spatial relationships.

GCN: All the above models only care about users' property information, which may result in a high false-positive rate. However, GCN applies a graph structure to capture the correlation between users and entities. GCN is an extension of CNN, which is a class of neural networks for learning graphic representations of data, such as node features and structural information. Jiang et al. [42] used a graph structure to model the correlations among users, for example, communicating by email or working on the same host. They also combined user behaviors, user profile information, and entity properties into feature vectors of nodes. They then input them to the convolutional network and chose cross-entropy as the objective function to predict anomaly users (nodes) in the graph and their related malicious groups. Li et al. [43] presented a Dual-Domain Graph Convolutional Network (DD-GCN) to fuse user behavior information from the feature domain and topology domain into a high-level representation. They applied a weighted feature similarity mechanism function to develop heterogeneous graphs based on the original users' relationship information and their activity feature similarity. They then extracted specific graph embeddings from the original topology domain and graph concurrently. Next, the two types of embeddings were fed into an attention mechanism to learn the importance weights of the user's embeddings in the two domains and fused. Finally, the output module classified the output to complete insider threat detection tasks.

The GCN is not required to label all the nodes in the graph, and thus, partly labeled data can be used to train the model. Only the connection relationships between the users capture the structural information of the network. Thus, structural information seems to carry less weight in improving the accuracy of insider threat detection. As a result, the GCN's performance lags behind that of the RNN when it is fed with a large imbalanced dataset. Additionally, it is not easy to implement because it requires prior knowledge and significant effort to construct a complex graph structure to learn the topology as the

input of the model.

From Table 2.2, we can see that a Deep Feedforward Neural Network (FNN) is not ideal for modeling time-series data because it lacks mechanisms to handle sequential or time-dependent patterns. FNNs process inputs independently without memory, making it hard to capture temporal dependencies or trends over time. They also require fixed-size input vectors, which limits their ability to handle variable-length sequences. Even when reshaped, FNNs cannot inherently understand sequence or handle long-term context, making them unsuitable for tasks involving complex temporal relationships. The CNN model easily loses valuable information because of the pooling layer and requires a large dataset with labels. Thus, it is not suitable for modeling time sequence patterns. The GCN can model the behaviors of the users and their relationship, but its accuracy needs improvement and requires prior knowledge and effort to construct the graph. RNN is capable of handling sequential data but is limited by the slow training time and capturing the long-term sequence dependencies. The LSTM provides a fair solution to the long-range sequence dependency problem but disregards parallel computations; thus, the training is even slower than that of the RNN.

2.1.4 Transformer Methods

2.1.4.1 Transformer

The Transformer model is a type of deep learning model initially used for natural language processing (NLP) tasks. It was introduced by Vaswani et al. [44] in 2017. The core innovation of the Transformer model is the attention mechanism, which allows the model to weigh the importance of different words in a sentence. Each word in a sentence is compared with every other word, and the model generates a set of attention scores. This allows the model to focus on relevant words when encoding a particular word, capturing the global context and relationships between words more effectively. In addition, unlike traditional sequence models like Recurrent Neural Networks (RNNs), the trans-

Table 2.2: The Deep Learning Methods of Insider Threat Detection

Approaches	**Pros**	**Cons**	**Reference**
DEEP BELIEF NETWORK	It is suitable for multidomain feature processing	It cannot handle temporal data and generate lots of False alarms	[33]
AUTOENCODER	Autonomous organization level offering few false alarms	It cannot handle temporal data; late data fusion loses the early chance of modeling correlations across domains	[34]
RNN	It can efficiently capture temporal pattern in data	Slow and expensive training; tend to forget long term dependency; it can find suspicious event but cannot ensure if it is insider threat	[35] [36] [37] [38] [39] [40]
CNN-LSTM	It can efficiently capture temporal patterns in data; it also enables faster processing of matrices by using CNN.	Slow and expensive training; easily loses much valuable information due to pooling layer and need a large labeled dataset	[41]
GCN	It can model the correlation between users and users' behavior	It is difficult to implement; it requires prior expert knowledge and much effort to construct the graph	[42] [43]

former does not employ recursive or convolutional mechanisms, resulting in shorter paths between input and output positions within the network. Moreover, positional encodings are added to the input embeddings to provide the model with information about the position of each word in the sequence. It allows the model to capture long-term patterns regardless of their distance in the sequence, which is particularly useful for understanding complex language structures and avoids the issue of gradient disappearance. Due to these characteristics, the transformer model can process entire sequences in parallel simultaneously, enhancing both training and inference efficiency. Furthermore, the transformer model requires fewer assumptions about the data's schema information and applies tokenization, making it a versatile architecture for processing diverse heterogeneous input data types. All these advantages have made the transformer model become the backbone of many state-of-the-art models and applications in NLP and beyond.

The original Transformer model consists of an encoder and a decoder. The encoder processes the input sequence and generates a set of encoded representations. The decoder uses these representations to generate the output sequence, one word at a time. The encoder is composed of 6 blocks, each with two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. Each sub-layer has a residual connection followed by layer normalization. The structure of the decoder is similar to the encoder, but with an additional multi-head attention mechanism that attends to the encoder's output.

Transformer model is widely applied in various Natural Language Understanding tasks, like text classification, sentiment analysis, and named entity recognition. It also used in a variety of text generation tasks, such as machine translation, text summarization, and conversational agents. Furthermore, transformer models have also been adapted for use in other domains, for example, computer vision (e.g., Vision Transformers) and speech processing.

On the other hand, the original Transformer model is designed primarily for sequence-to-sequence tasks like machine translation. Therefore, the transformer model is less

adaptable to other tasks without significant changes. Additionally, it relies on a unidirectional attention mechanism within the encoder and a combination of self-attention and encoder-decoder attention in the decoder. That is to say, it only can capture context from one direction, Thus, it may not understand nuanced meanings in sentences. Moreover, the training of the original Transformer is typically task-specific without a pre-training phase. Furthermore, its architecture is relatively simpler compared with other language models as it only has a typical 6 layers for both encoder and decoder.

Wu et al. (2022) [45] utilizes a variant transformer model to detect external intrusion on network traffic data. Differing from the original Transformer model, an extra multi-headed self-attention sub-layer with masking is incorporated into each decoder. However, our experiment shows this change makes the performance of insider threat detection degrade. Therefore, this study explores from another direction: only use and further simply the encoder to fully take advantage of the language understanding capability to improve the performance of insider threat detection. The detail is in section 3.4.2

Transformer variant models can undergo pre-training on extensive datasets and then be fine-tuned for various downstream tasks. Wang et al.[12] integrate Digital Twin technology and self-attention mechanisms to analyze user behavior and entities in 2023, utilizing advanced data augmentation techniques such as contextual word embedding with BERT model and contextual sentence embedding with GPT-2 model to address data imbalances. The paper proposes DistilledTrans, a simplified transformer model, and evaluates its performance alongside other transformer models and hybrid models, including BERT, RoBERTa, BERT+ CNN, RoBERTa + CNN, BERT + LSTM, and RoBERTa + LSTM networks. Experimental results demonstrate the superior performance of DistilledTrans on sporadic datasets CERT r6.2. In contrast, pre-trained models like BERT and RoBERTa exhibit high performance on dense datasets CERT r4.2, suggesting that complex hybrid methods combining transformer and other deep learning models are not necessary. Overall, the proposed framework demonstrates significant advancements in insider threat detection with improved accuracy, efficiency, and interpretability compared

to existing models. However, that solution is based on centralized learning and does not involve FL, PETuning, or TL.

2.1.4.2 BERT

Bidirectional Encoder Representations from Transformers(BERT) model [8], developed by Google, is a pre-trained transformer-based model. It was trained on a huge amount of unlabeled data, including the Books Corpus which has 800 million words and English Wikipedia which has 2,500 million words. BERT underwent pre-training on tasks such as masked language prediction and next sentence forecasting. We chose the BERT-based model as one of the foundational models. This model consists of 12 encoders with approximately 110 million parameters. It contains 12 bidirectional self-attention heads and 768 hidden layers

2.1.4.3 XLNet

By relying on corrupting the input with masks, BERT overlooks the reliance between the masked positions, resulting in a divergence between the pre-trained model and the fine-tuned model. To address these shortcomings, **eXtreme Language understanding NETwork(XLNet)** [10] was proposed as a generalized autoregressive pre-training method. XLNet has two key mechanisms: (1) **Bidirectional Context Learning**: XLNet maximizes the expected probability over all permutations of the factoring order, enabling it to learn bidirectional contexts. It employs a permutation-based approach named Permutation Language Modeling, allowing the model to learn from all possible combinations of input tokens rather than a fixed order. This is achieved by training the model to predict the probability of a token given all other tokens in the input sequence, irrespective of their position. (2) **Long Term Dependency**: XLNet overcomes BERT's shortcomings by using its autoregressive attribute. It integrates ideas from Transformer-XL[36], an advanced autoregressive model, into pre-training to get long-term reliance in the input sequence. This is accomplished through a segment-level recurrence mecha-

nism, enabling the model to preserve the memory of the previous segment while handling the current segment. Under equivalent experimental settings, XLNet consistently beats BERT across a range of 20 tasks, achieving notably superior results. These tasks encompass text classification, question answering, natural language inference, sentiment analysis, and document ranking.

2.1.4.4 RoBERTa

Robustly Optimized BERT Approach(RoBERTa) [9], developed by Facebook AI, shares similarities with the BERT model but introduces several modifications. Like BERT, RoBERTa learns contextualized word embeddings using a self-attention structure. However, it eliminates the Next Sentence Prediction (NSP) objective. RoBERTa is trained on a larger dataset (160GB), including the CC-NEWS, OPENWEBTEXT, and STORIES datasets. Additionally, RoBERTa increases the batch size, sequence length, and learning rates. It also employs dynamic masking strategies during training to enhance the robustness and generality of word embeddings. Furthermore, RoBERTa adopts byte-level Byte Paired Encoding (BPE) with a vocabulary of 50K sub-word units as a tokenizer, replacing the character-level BPE with a vocabulary of 30K sub-word units used by BERT.

2.1.4.5 DistilBERT

The above-mentioned models are too large, slow, and heavy to fit for edge applications. **DistilBERT** [11] was proposed to pre-trained a small, fast, and light general-purpose language representation model. On the other hand, this model can then be fine-tuned to achieve strong performance across a wide array of tasks, akin to its larger equivalents. The knowledge of distillation is applied to simply the model during the pre-training phase. The concept suggests that once a large language model has undergone training, it's possible to approximate its full output distributions using a smaller network. The experiments demonstrated the potential of reducing the size of a BERT model by 40%,

computing 60% quicker while keeping 97% of its language understanding abilities. To harness the inductive biases learned by larger models during pre-training, a triple loss integrating language modeling, distillation, and cosine-distance losses was introduced. It is more cost-effective to be pre-trained, and suitable for edge device computations or resource-constrained environments.

2.1.4.6 GPT

Generative pre-trained transformers (GPT) is a family of pre-trained varied transformer models for generative language learning, which was proposed by OpenAI in 2018 [46]. Different from the above pre-trained large language models built for understanding and analyzing text, the uniqueness of GPT is its ability to generate coherent and contextually appropriate text given a prompt or input because it is designed for autoregressive text generation tasks:

1. **Transformer Decoder:** GPT only adopts Transformer decoders that process input in a unidirectional way. This means it looks at tokens sequentially from left to right in one direction. This architecture allows the model to handle long-range dependencies and contextual information in text more effectively than previous models, like CNNs or RNNs.

2. **Self-attention Mechanism:** GPT employs causal (or masked) self-attention to predict the next token in a sequence given the previous token. This means that during training and inference, each token in the sequence attends only to tokens that precede it in the sequence, preventing it from looking ahead to future tokens during training and generation to ensure aggressiveness

3. **Generative Capabilities:** This autoregressive training approach teaches GPT to extend text sequences logically and thus helps GPT generate coherent and contextually relevant text.

4. **Training Procedure:** The model is first trained on a large corpus of text data in an unsupervised method, learning to predict the next token in a sequence. This helps the model understand language structure, syntax, and general knowledge. After pre-training, GPT can be fine-tuned on specific datasets with labels for particular tasks.

The GPT model has undergone several iterations, with each version improving upon its predecessor in terms of scale, capability, and performance due to their improved capacity to learn from data:

1. **GPT-1:** Introduced in 2018, GPT-1 is a 12-level transformer decoder with 12 heads, and a linear layer and a softmax layer are on top of those with 117 million parameters in total. It was pre-trained on 4.5 GB text from BookCorpus. It demonstrated the potential of pre-trained language models for various NLP tasks.

2. **GPT-2:** Released in 2019, GPT-2 is a scaling up of GPT1 that shares the same architecture as GPT-1, but with modified normalization. It is pre-trained on 1.5 billion web text, including 40GB of text, and has 1.5 billion parameters. It demonstrated improved text generation capabilities and was able to perform reasonably well on tasks it was not explicitly trained for, showcasing the power of transfer learning.

3. **GPT-3**: Launched in 2020, GPT-3 has 175 billion parameters, making it one of the largest and most powerful language models at the time. It excels in few-shot and zero-shot learning, where it can perform tasks with minimal to no task-specific data.

4. **GPT-4**: The latest iteration cutoff in 2023, GPT-4 continues to build on the success of its predecessors with more parameters, offering even more sophisticated language understanding and generation capabilities.

The previous language models excel in tasks that demand deep understanding and contextual comprehension, while GPT is ideal for tasks that require generating text based

on a prompt, such as chatbots, story generation, creative writing, and text completion. GPT models represent a significant advancement in the field of natural language processing, leveraging the transformer architecture to provide powerful text generation and understanding capabilities. GPT-2 is chosen by this study to predict sentence embedding based on the context it pre-learned to augment the minority class in the experiment dataset.

2.2 Digital Twin

The Digital Twin (DT) concept was introduced by Grieves et al. [47] in 2002 as an integrated probabilistic simulation of a physical asset or system to emulate the lifetime of its matching physical twin by using the best existing physical model, history information, latest sensor data updates, etc. DT was renewed by El Saddik and Abdulmotaleb [48] in 2018 as a digital replica of a living or non-living physical entity. DT synchronizes with the physical entity through smooth data communication. In contrast to many industrial or manufacturing DT applications that often mimic a component of the industry system or the whole industry system, a digital twin in the context of this study is a digital counterpart of an actual real-world physical insider, or an entity such as a computer or device, and its relationship with other digital twins. This is used to simulate the behavior of a specific user in an organization.

2.3 Federated Learning

In 2016, Google pioneered the introduction of Federated Learning (FL) [7] to facilitate efficient learning across multiple participants while ensuring data privacy and security during data exchange. Over the past few years, FL has experienced rapid growth across various industries and applications. In FL, each participant conducts local training using private data and subsequently uploads the gradient or weight update to the server. These updates from all users are then aggregated to refine the global model, which is subse-

quently distributed back to all clients for the next iteration. FL effectively solves isolated data island issues to construct a robust global model while safeguarding personal data privacy and information security. However, FL encounters challenges such as high communication costs and diverse client data distributions. FedSGD is introduced alongside the FL concept. It entails the computation of a weighted average of the local gradient. While Stochastic Gradient Descent (SGD) serves as a commonly employed optimization function in Neural Networks (NNs), the process of aggregating gradients at each epoch results in considerable costs concerning bandwidth consumption and computing power. To tackle this challenge, FedAvg is proposed to aggregate model parameters rather than gradients, thereby permitting each client to train the model locally and share updates after several epochs. Our study adopts this approach.

In the realm of insider threat detection, Amiri-Zarandi et al.[27] introduce a federated learning system that employs the AutoEncoder model to improve privacy and Shapley value to enhance explainability. However, it lacks implementation details and in-depth analysis. It neglects the requirements for localizing models and maintaining global performance, reducing communication cost and storage overhead, overcoming data imbalance, and detecting new or unknown attacks. The solution is not an end-to-end solution but needs human efforts to investigate at least 20% of the data. The performance is not objective as it relies on the threshold set by humans. Additionally, it does not involve any data processing, powerful detection models, PETuning methods, and effective TL. Moreover, the performance metrics are incomplete and the published performance cannot compete with FedITD proposed by us. Furthermore, the Shapley value is not suitable for temporal sequence data which are often the input data of learning models capturing temporal patterns.

Qu et al. [49] are pioneers in applying Transformers to FL, demonstrating that Transformers exhibit greater effectiveness and robustness in handling client heterogeneity when compared to conventional architectures like Convolutional Neural Networks (CNNs). Weller et al. (2022) [50] provided experimental evidence demonstrating that

Pre-trained LLMs can mitigate adverse effects from non-IID and diminish the gap in accuracy between them and centralized learning. Nevertheless, the substantial communication cost in FL systems results in slow and impractical FL for real jobs. Moreover, LLMs may present challenges for local clients with constrained hardware abilities for computation, memory, and storage. Furthermore, they do not apply it in insider threat detection but only apply to NLP tasks.

2.4 Parameter-efficient tuning (PETuning)

Parameter-efficient tuning (PETuning)method aims to retain the majority of parameters in LLMs frozen while fine-tuning only lightweight extra parameters or a small subset of the parameters for specific downstream tasks. This approach allows Pre-trained LLMs to leverage existing knowledge, minimize resource requirements (computation, memory, storage, etc.), and alleviate communication overhead in FL, which is predominantly influenced by the size of model update parameters. Houlsby et al. [13] developed a down-up projection module named Adapter within each transformer layer for BERT in 2019. Another approach proposed by Zaken et al. involving fine-tuning the backbone of the pre-trained model is Bitfit [15] which focuses solely on adapting the bias term of the network. Hu et al. [14] demonstrate the feasibility of training large glossary language models while maintaining accuracy and privacy through the introduction of low-rank adaptation (LoRA) in 2021. Nevertheless, it's worth noting that all of these methods were trained centrally on the server without taking into account user privacy concerns, under the assumption of having access to the source data.

Sun et al. [51] introduced the FedPEFT framework in 2022, which incorporates three PETuning methods (Bias, Adapter, Prompt) from pre-trained visual models into FL. Their findings indicate that employing featherweight PETuning methods in FL can notably alleviate the communication burden while preserving performance, showing superior performance across various FL settings. Additionally, Chen et al. [52] expanded

PETuning techniques to visual language models within FL in 2022, demonstrating that PETuning can accelerate convergence rates. Through extensive experiments, they examine three fine-tuning methods (prompt, Adapter, and Bitfit) across two types of pre-trained models (vision-language models and vision models) for FL. Key findings from their experiments include: Fine-tuning the bias term of the backbone performs optimally when utilizing a robust pre-trained model; vision-language models (e.g., CLIP) surpass pure vision models (e.g., ViT) and exhibit greater resilience in few-shot settings; FL with pre-trained models achieves higher accuracy compared to pure local training, as it mitigates overfitting issues. However, these studies overlook the significant issue of privacy attacks in FL. Zhang et al. (2023) [16] address the critical issue of privacy attacks inherent in FL. They introduced FedPETuning to evaluate the effectiveness of RoBERTa model in natural language understanding, focusing on the GLUE datasets. FedPETuning maintains satisfactory performance, achieving over 95% of FedFT's performance, while significantly reducing communication overhead. This approach effectively defends against data reconstruction attacks, which aim to recover text from gradients or weight updates uploaded by clients. Since FedPETuning only communicates a small part of the entire model parameters with the server during FL, it becomes impractical for attackers to reconstruct the original text from such lightweight model parameters. However, it is apparent that the samples stored on the server exhibit significantly varied probability distributions compared to the data generated by each client. As a result, the global model fails to achieve localization on each client's site. However, these studies overlook the critical issue of how to build the custom model for each client after FL. Furthermore, none of these studies examine the impact of PETuning methods using various foundation models in FL that are essential for deploying pre-trained models in FL scenarios.

2.5 Transfer Learning

Transfer learning is a machine learning technique where a model developed for a particular task is reused as the starting point for a model on a second, related task. This approach leverages the knowledge gained while solving the initial task (source task) to improve learning and performance on the new task (target task). Transfer learning is particularly useful when the target task has limited labeled data, as it allows the model to benefit from the larger, labeled dataset of the source task. This study utilizes **Unsupervised domain adaptation (UDA)** method. UDA is a transfer learning technique used to adapt a model trained on a labeled dataset from a source domain to a target domain where labeled data is not available. This approach is essential when the data in the target domain differ from the source domain in significant ways. Source Domain is the domain with labeled data, which is used to initially train the model while target Domain is the domain with unlabeled data, where the model is intended to be applied. UDA aims to bridge the gap between different domains by employing various techniques to align feature representations and improve model performance in the target domain.

When utilizing the global model trained by the FL system directly, it often performs inadequately on specific client data. This discrepancy arises from the distribution disparity between the client and the globally trained data. Continuing full-tuning or PETuning on client data fails to yield satisfactory results. Instead of involving more parameters of the model in training, transfer learning emerges as a solution. Transfer learning facilitates the transfer of knowledge from existing domains to a new domain without necessitating model reconstruction. It effectively addresses the challenge of training models when data is limited, such as in the case of small data islands. Chen et al. introduce FedHealth [53], the pioneering federated transfer learning framework for wearable healthcare, aimed at classifying human activities. Moreover, IoTDefender [54] employs deep domain adaptation to develop personalized Intrusion Detection System (IDS) models for attack

detection in 5G IoT networks. This approach tackles distribution differences between traditional networks and IoT environments. However, these papers do not involve any PETuning method of LLMs in the FL environment and how to apply TL for LLMs in insider threat detection.

Sun et al. [55] introduced an easy and effective method for unsupervised domain adaptation called CORrelation ALignment (CORAL) in 2016. The CORAL method aligns the second-order statistics of the target and source's data distributions through a linear transformation without requiring any label in the target domain. Extensive testing on standard benchmark datasets showcases CORAL's exceptional performance. However, it is worth noting that CORAL relies on a linear transformation and thus it is not end-to-end. It involves a multi-step process wherein features are initially extracted, followed by the transformation, and subsequently training a machine learning model. Sun et al. [56] extended CORAL in 2016 by incorporating it directly into deep neural networks by introducing a new differentiable loss function named CORAL loss. This loss function minimizes the disparity between the learned feature covariance of the target and source domains. Compared to the original CORAL approach, the Deep CORAL method learns a more potent non-linear transformation that is much simpler to optimize. Furthermore, it seamlessly integrates with deep Convolutional Neural Networks (CNNs), exhibiting stronger performance on standard benchmark datasets. However, it has not been integrated into pre-trained LLMs and applied to temporal sequences yet. Additionally, they do not disclose how to get the best performance across the source domain, target domain, and global domain. Moreover, they do not verify its extensive capability on highly heterogeneous target data domains.

Chapter 3

DTITD: Insider Threat Detection Framework Based on Central Learning

In this chapter, we present and implement our proposed intelligent insider threat detection framework DTITD based on Digital Twin and self-attention based deep learning models. We first give an overview of the proposed framework in section 3.1. The data preparation module is introduced in section 3.2. After that, the model construction module is presented in section 3.3. The insider threat detection module is shown in section 3.4.

3.1 Introduction

Insider threat detection is a continuous process that requires frequent data processing, data augmentation, model training, model retraining and fine-tuning, model validation and testing, and model inference throughout its lifecycle. In addition, it often involves huge amounts of user activity and user profile information that are sensitive and private. This information cannot be accessed by unauthorized people. Therefore, this study proposes DTITS, an intelligent framework for Insider Threat Detection based on DT and Self-attention Based Deep Learning Models to timely and continuously monitor the

insider risk profile of an organization and take remediation actions. Adopting DTITD framework brings about the following benefits:

1. The application of digital twin technique in insider threat detection allows the corresponding physical users and entity's entire lifecycle and their correlations to be modeled, simulated, and regularly synchronized with the corresponding physical twins. With the power of DT graphs and DT's properties updated by user activity and user profile information in real-time, users' current behaviors and their changes over time are mimicked. Thus, we can easily obtain an accurate broad overview and in-depth views of the insider risk profile at the organizational level and quickly take measures to address the risks.

2. DTITD is also an advanced AI solution for detecting insider threats in real time. It allows heterogeneous data from various sources to be quickly integrated, processed, and fed into downstream deep learning models to detect insider threats. DTITD is a good implementation of User and Entity Behavior Analytics (UEBA). UEBA is a type of cyber security process that not only models the regular behavior of users and entities but also associates user behavior with entity behavior. The term "entity" in the context of cybersecurity can refer to machines such as PC, networked devices, servers within the network, or IT systems. Thus, the UEBA follows the behaviors of all the users and entities in an organization, rather than passively monitor devices or respond to security events. A normal user profile is the user's activity sequences and frequencies, which are shown on a usual basis. By modeling such baselines, the UEBA can detect any abnormal actions when they are different from normal user profiles. Therefore, it can more accurately detect dubious behavior, likely threats, and attacks that traditional security systems may not detect especially new or "unknown unknowns," risks that conventional detection methods are neither aware of nor understand.

3. This framework not only enhances cybersecurity but also aligns with broader Envi-

ronmental, Social, and Governance (ESG) goals, contributing to a more sustainable, responsible, and ethical business environment. It provides transparency and interpretability to insider threat detection and response processes and thus improves corporate governance practices. Meanwhile, it safeguards sensitive data, protects individuals' privacy, and complies with data protection regulations.

3.2 Proposed Framework

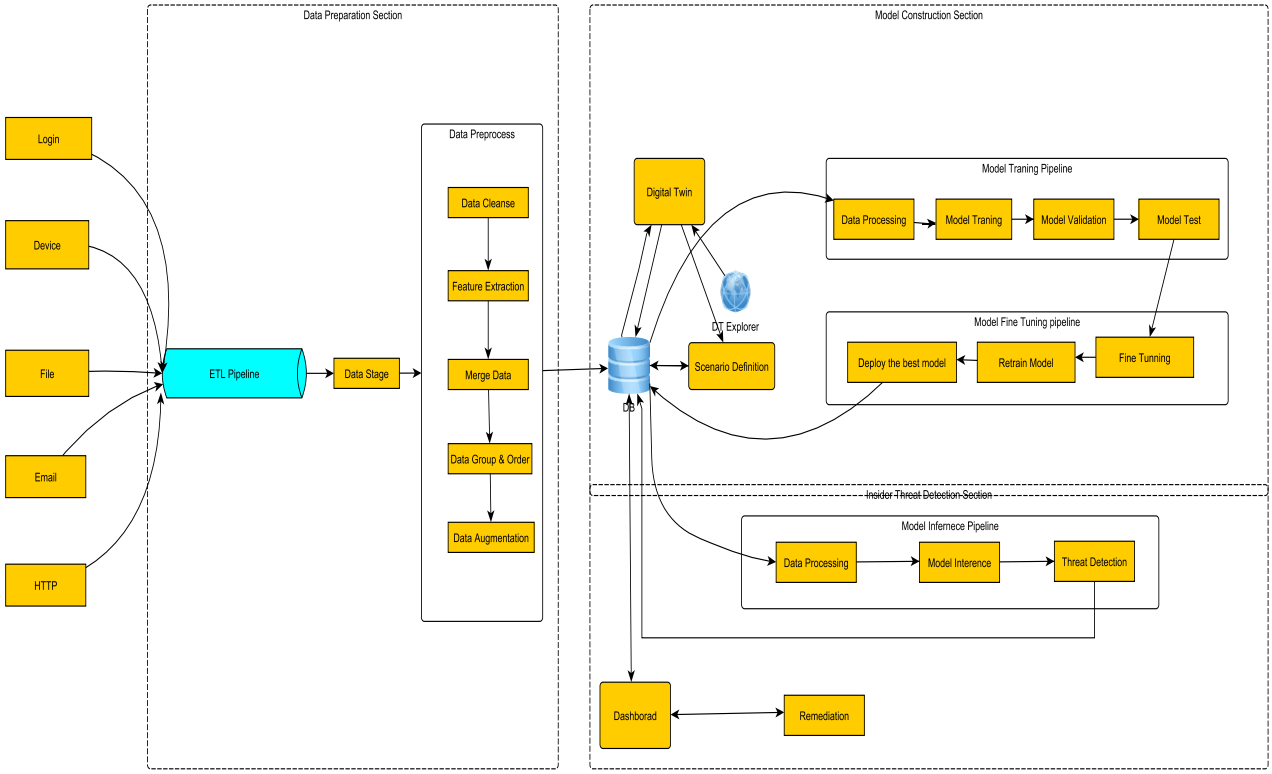

Figure 3.1: The system architecture of the proposed DTITD Framework.

Fig 3.1 provides an overview of the DTITD framework. The DTITD comprises three sections: data preparation section, model construction section, and insider threat detection section. The ETL pipeline collects and filters various real-time log data, including login, device, file, email, and HTTP log data, and then moves them from the data sources on-premise to the data stage on the cloud. Raw log data are preprocessed in four phases:

data cleaning, data merging, feature selection, data group, and order. Subsequently, pre-processed data will persist in the database and update the properties of the digital twin. The scenario definition defines one scenario by filling in the desired user-defined parameters. Then, it triggers the model training pipeline so that data are further processed, and the deep learning model learns the scenario's user behavior from the historical data. Next, the super-parameters of the models are fine-tuned to compare performance and select the best model. After that, the model inference pipeline is triggered and periodically run so that real-time data from logs can be processed in a timely manner, and the trained model will be run to detect insider threats. The insider threat detection results are sent back to an associated result table in the DB. If there are abnormal users or activities, the system will create an incident to notify the dashboard and update the DT graph. The dashboard will show the detection results. Security analysts can take remediation actions to revert the possible damages done or prevent the incident from happening again.

3.2.1 Data Preparation Section

Data Preparation Section is responsible for collecting data from various data sources and performing data preprocessing including data augmentation. DTITS integrates data from various data sources through Kafka ETL pipeline and moves data to the data stage on the cloud. Then it performs data preprocessing including data cleansing, feature extraction, data merge, data group, and order to generate user behavior time sequence. Next, it makes data augmentation for the imbalanced data set. The data augmentation methods in this chapter include context word embedding insert and substitution by using pre-trained transformer model BERT and context sentence embedding by using pre-trained transformer model GPT2.

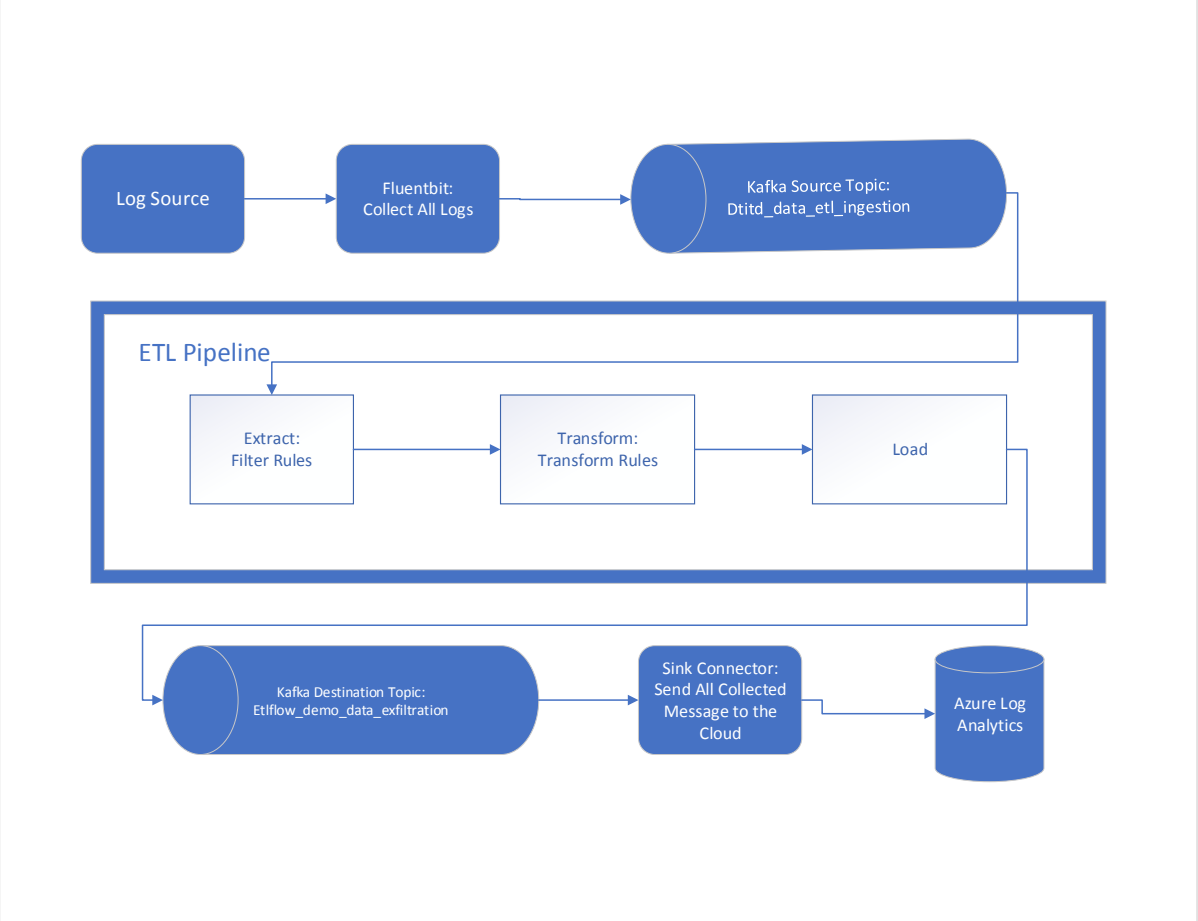

Figure 3.2: The Architecture of the ETL Pipeline for insider threat detection.

3.2.1.1 ETL Pipeline

Our data include user behavior data and user profile data. The former comes from various logging systems e.g., host logs, network traffic log, emails, file access logs, Web browsing log, firewall logs, etc. These data are all from real-time data sources that often need to be captured and processed in a timely manner and converted to time sequences so that we can quickly detect and respond to take remediation measures. While the latter comes from Active Directory or HR database e.g., employees' identities, roles, titles, teams, projects, departments, supervisors, relationships to other users, and psychological data, etc. As Fig. 3.2 shows, initially, Fluentbit collects the real log data from data source systems and moves to Kafka pipeline source topic. Then, Kafka pipeline filters the original data flow to get the data that we are interested in and transforms the data to our desired data format as per filter rules and transformation rules defined in configuration files. Next, we send the filtered and converted data to the data stage (Azure Log Analytics Workspace) on the cloud platform through Sink Connectors.

3.2.1.2 Data Preprocessing Module

As Figure 3.3 shows, the Data preprocessing module includes data cleansing, feature extraction, data merging, data grouping and sorting, and data augmentation. We perform data cleansing to eliminate dirty data and fill up missed values, extract and generate useful features from cleaned data such as user ID, activity time, activity types, etc. Combine data from five domains: login, device, file, email, and http, group the data by each user and each day, and sort each user's behavior by occurrence time to form a user behavior sequence for each day. It can also augment the minority classification's data for a highly imbalanced dataset. This chapter applied context embedding insert and substitution by using a pre-trained transformer model BERT and context sentence embedding by using a pre-trained transformer model GPT2.

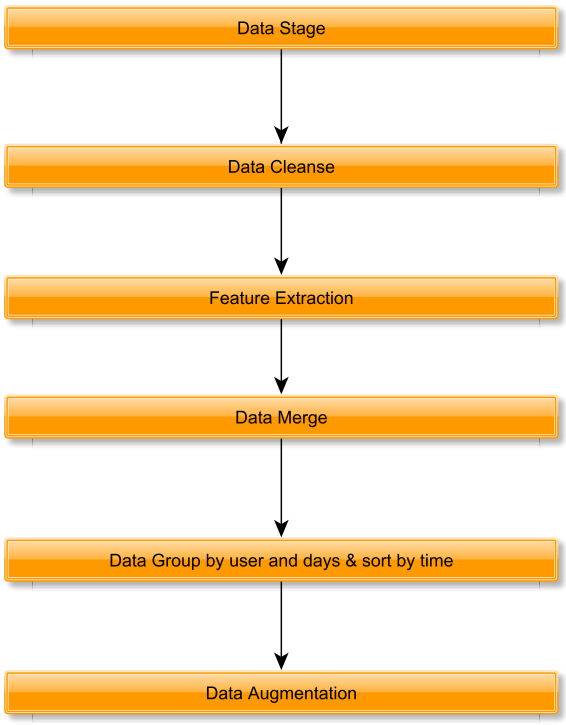

Figure 3.3: The Data Preprocessing Module

3.2.1.3 Data Stage

Data Stage is the data stage tables in Azure log analytics workspace. The Azure log analytics workspace is a data service that has reliable and scalable abilities to persist collected and preprocessed data with strong log management capabilities. Through the Sink link, the data are fed into the data stage tables in the Azure log analytics workspace.

3.2.2 Model Construction Section

Model Construction Section is responsible for constructing custom models, fine-tuning pre-trained models, and managing DT services. It consists of Digital Twin Service, DB, Scenario Definition, custom model building (mode training, validation, and test) module, and pre-trained models' fine-tuning module. DT timely displays and monitors the current uses' and entities' status and correlations.

3.2.2.1 Digital Twins Module

The Digital Twins service instance is a platform as a service (PasS) storing digital DT models and DT graphs with their states, and orchestrated event processing on the cloud. DT graph is a graphic representation of the DTs of users and entities and their relationships. Users are represented by circular nodes, whereas entities are represented by rectangular nodes. User DTs' properties contain rich user profile information, such as IP, port, and normal work hours. Various edges capture the structural information and interactions occurring among nodes, for example, assigned PCs, teammate relationships, same role relationships, teammate relationships, connections on the same device, or email communication. The file, Email, and Http web browse features can also be incorporated into the DT graph. DT Explorer is a client application that allows users to quickly create DT models, manage DT models and graphs, and run queries to find existing DTs with simple drag-and-drop actions. Security analysts can also explore insights from existing DTs or share them on the DT service platform using the DT query API. DT services can generate events to trigger external computing resources handling business logic and data processing. This module is achieved by using Azure Digital Twin Services.

For example, Fig. 3.4 shows that IT administrator PLJ1771 has an assigned PC, PC7272. He has team members DNS1578 and DMC1766, who have the same role as administrators. He also has email communications with user SDW0270 and JDS0516 who are managers in other departments. He was connected to PC0856, PC6326, PC6760, and PC7272. This is normal behavior since he performed his daily work on these PCs. However, he connected to his supervisor's machine PC3999 by coping with a keylog file to that machine and logging on to that machine. This behavior is very suspicious, and thus the user's status is changed to red.

Digital Twin's work process is like this: firstly, create a client connection to the DT instance using its URL endpoint. Then create or upload DT models using Digital Twin Model Definition Language, making sure to include the properties for the time-series and the graph relationships. Next, build digital twins for the nodes in the DT graph,

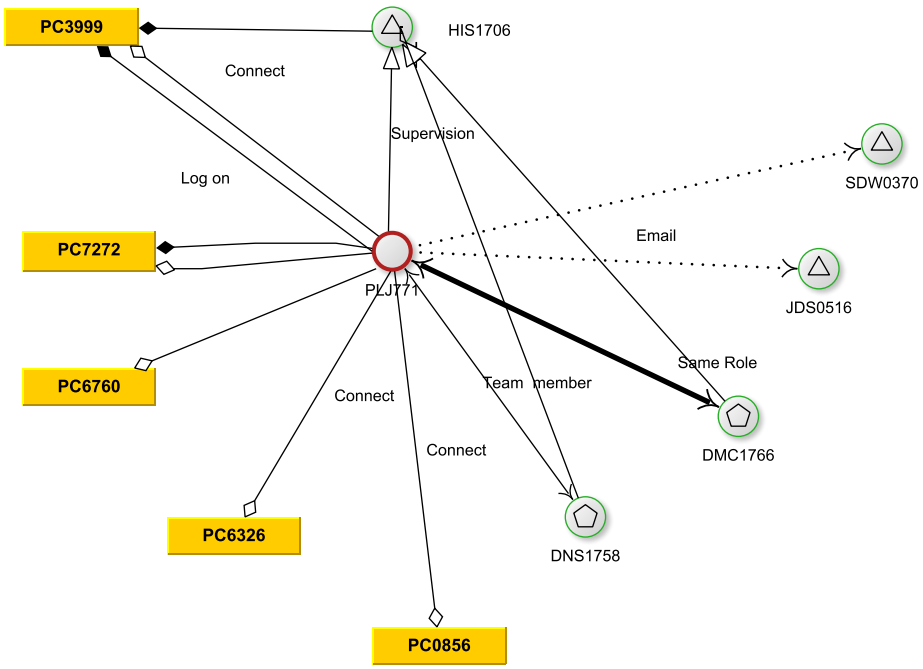

Figure 3.4: DT Graph of the User PLJ1771.

by referencing the correct model ids, and initializing the properties. After that, add relationships to the DT graph using the defined topology. Finally, update properties by using real-time incoming data including time series data and user's category information.

3.2.2.2 Scenario Definition

Scenario Definition Module is to define one scenario by filling in the necessary user-defined parameters. It includes querying the twin graph to get a scenario's required context information e.g., Digital Twin ID entries, selecting the target properties in the scenario, e.g., the behavior and category features, from the digital twin graph, and selecting data source information in DB, e.g., table name, column mapping etc.

3.2.2.3 Data Storage

Data Storage is implemented by using Azure SQL DB. An Azure Logic App runs a Kusto Query Language (KQL) every 15 minutes to extract new fed data, convert the

data into a normalized relational schema, and load them into the Azure SQL DB for insider threat detection. The DB stores the pre-processed training dataset for model training. Moreover, DB also stores the DT models, trained ML model metadata, and prediction results. User activity data are automatically extracted from the Azure Log Analytics Workspace and archived in the staging tables in the DB. DB runs a scheduled script to update digital twin properties by using Azure Functions through the event hub. Prediction result tables are created for every scenario, and take the name of the scenario. The prediction results include a boolean value "isInsiderThreat" indicating the prediction result, severity of the anomaly, anomaly score, feature's importance reflecting each feature's contribution to the prediction results, and other context information sent back form the ML inference pipeline to DB. In addition, we can directly upload data to a DB table by importing data from local files

3.2.2.4 Model Training Pipeline

ML Model Training pipeline takes in preprocessed data to train, validate, and deploy the ML model. It often involves the following steps:

1. Move the training dataset from Azure SQL DBto Azure Blog Storage
2. Create the Run Configuration defining the environment needed to run the pipelines on the compute target
3. Create the Data Preparation Pipeline Step where we run Python script to retrieve the raw input data from Azure blob storage, process the input data, and output the processed data that will be used in the subsequent steps.
4. Create the Model Training Pipeline Step where we use the processed data as the model input to train the models
5. Crate the Model Validation Step where we validate the model using validation data sets.

6. Create the Model Deployment Step where we register the model and deploy it to the Web service endpoint.

7. Create an experiment. Submit the pipeline to run and monitor the running pipeline

3.2.2.5 Model Fine-tuning Pipeline

Model Fine-tuning Pipeline loads the pre-trained model, fine-tunes super-parameters of the model, and retrains the model to obtain the best model. Finally, deploy the best model.

3.2.2.6 Python Scripts

Python scripts are running in Docker containers in Azure Container Registry and performing the following functionalities:

- Process the raw data and convert data to the format desired by the deep learning model
- Call Azure API to run the steps in the model training pipeline or model inference pipeline
- In each step of the model training pipeline or model inference pipeline, python scripts will be run to read the relevant input data, train models, generate predictions, and store the results in DB.
- Visualize and interpret insider threat results

The configuration parameters are saved in the json file that can easily be reused, modified, or extended by users to meet the requirements of various scenarios.

3.2.2.7 Trigger

The ML model training pipeline and the ML model inference pipeline can be triggered using either the Azure Logic App or Azure Machine Learning Pipeline Schedule.

- **Azure Logic App**: Azure Logic Apps is a cloud-based service provided by Microsoft Azure that allows users to create and run automated workflows that integrate apps, data, services, and systems. For example, when the Logic App detects that a data file has been added to the container, it automatically triggers the ML model inference pipeline. The advantage is that it is an easy way to build automated workflows and integrate disparate systems and services. It reduces the need for custom code, accelerates development, and simplifies maintenance.

- **Pipeline Schedule**: Azure ML pipeline scheduling is a feature in Azure Machine Learning that allows users to automate the execution of their ML workflows at specific intervals or in response to certain events. For instance, once an ML model training or model inference pipeline is published, a Schedule can be used to submit the pipeline at a specified interval. Using scheduled pipelines, you can ensure consistent execution of your ML tasks, improve efficiency, and reduce the need for manual intervention. This is particularly beneficial for tasks that require regular updates

3.2.3 Insider Threat Detection Section

Insider Threat Detection Section is responsible for performing insider threat detection tasks. It consists of Model Inference Pipeline, Alert, Incident Management, Dashboard, and remediation module.

3.2.3.1 Model Inference Pipeline

Once model training is done, a logic app or pipeline schedule will trigger the model inference pipeline to apply the trained and registered model to make the prediction to detect insider threats. The model Inference pipeline consists of two steps: 1) the Data Prep Pipeline Step processing the new batch input data; 2) the Inference Pipeline Step running the trained model and explainer to make inferences on the newly processed

data. Input data is periodically uploaded in bulk into the Azure blob storage from DB. A schedule or logic app monitors changes to the blob storage and repeatably invokes the Model Inference pipeline if any data is added to the blob storage. This pipeline is developed using Azure Machine Learning Studio. Fig. 3.5 illustrates this workflow:

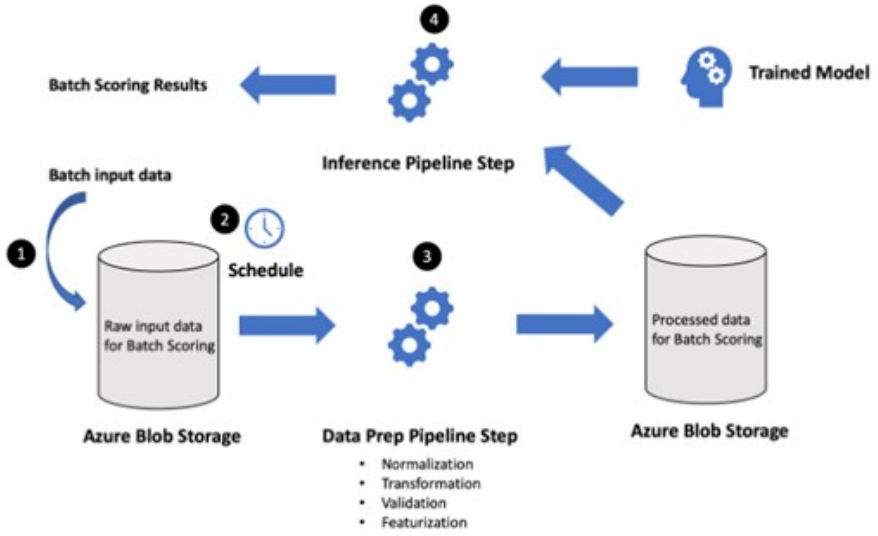

Figure 3.5: The Workflow of Model Inference Pipeline.

1. New input data is uploaded to blob storage from DB
2. The schedule or logic app triggers the Batch Inference pipeline
3. The model Inference pipeline runs the Data Prep step to tokenize the data
4. The model Inference pipeline runs the Inference step to detect insider threats

3.2.3.2 Alert, Incident Management, and Dashboard

This module is responsible for alerts, incident management, and data visualization. A scheduled query runs every 5 minutes on the prediction table in DB. When it finds there are abnormal users or activities, it will create an incident, raise the alarm, and update DT graph. The dashboard will present an updated DT graph, all the security metrics, the alarm content, and related context information to the security staff. This module is implemented using the Azure Sentinel.

3.2.3.3 Remediation

This module acts as the control component (actuator) of the DT, conveying physical anomaly response to control user behavior and physical systems. Security analysts can use this module to analyze the alarm-related information, investigate the root cause, and take appropriate actions. One of their options is to trigger an automatic remediation workflow to revert the damages and prevent the incident from happening again. For example, if there is a non-system user who makes modifications to the service construction of the Kubernete cluster, i.e., adding or removing services, nodes, or pods. the user's activity will be flagged as an anomaly by the ML inference pipeline and then Azure Sentinel rules will generate an incident to raise an alarm. The security analyst can launch the remediation workflow to handle the anomaly. As shown in Fig. 3.6, after getting a request, the workflow will parse the incident information and provide it to the dashboard. The remediation actions that can be taken include:

1. Block the user so that they don't have access to the cluster anymore: Authorization is usually performed using Role-Based Access Control (RBAC) in Kubernetes clusters. In RBAC, a user (or group of users) is "bound" to a role through a "role-binding" entity. A role determines what actions are allowed on what resources. The security analyst will find what roles the user has and then delete a role binding in Kubernetes.

2. Revert Malicious Actions: Add back any service or node/pod that was removed by an attacker, delete the service or node/pod that was added by an attacker, or any other actions. After that, the workflow will send the email back to the security analyst.

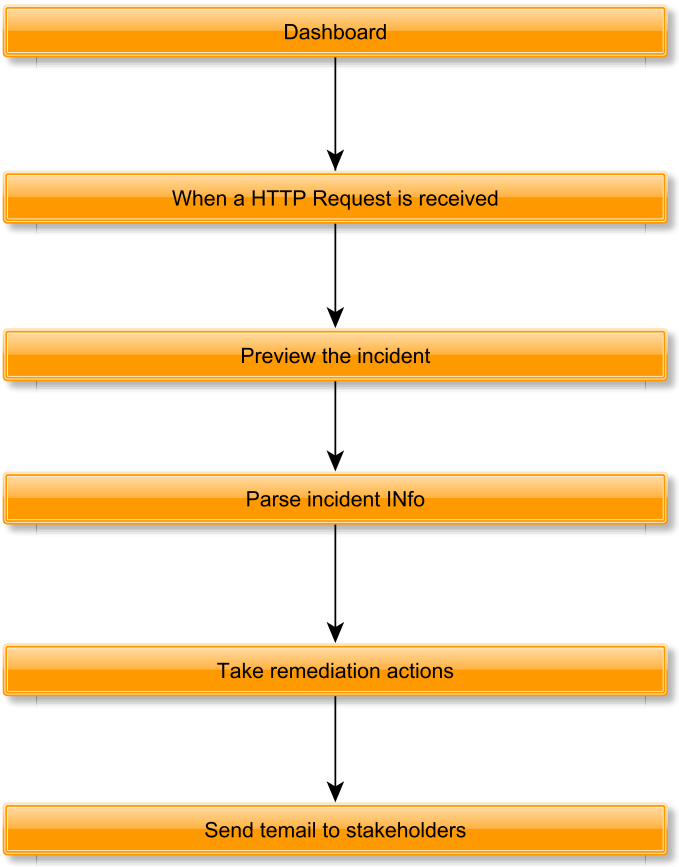

Figure 3.6: Remediation Workflow

3.3 Data Processing

3.3.1 Dataset

It is very difficult to get real insider threat data due to legal, privacy, and security reasons. The CERT dataset [57]is the most popular insider threat open dataset, which is a set of synthetic scenarios, malicious users, and their behavior data. It was developed by the CERT Division of the Software Engineering Institute at Carnegie Mellon University. It stimulates the IT-based activities and personal profiles of all employees in a synthetic organization named DTAA. From Fig 3.7, we can see that It consists of five log files: logon.csv archives all employees' logon and logoff actions on PCs; email.csv records all the email activities of employees; http.csv collects all the web browsing history records of employees; file.csv documents file-related operations (open, write, copy, or delete); and decive.csv maintains the connection information for the removable drive. In addition, the CERT dataset provides HR data for all employees in LDAP.csv and the psychometric score for all employees in psychometric.csv.

Data Files	Content	Attributes
LDAP.csv	• Record the list of employees of the month	• employee_name, • user_id • Email • role
device.csv	• Record the behavior of file access on the device	• Id, user • date • Pc • activity
email.csv	• Records the email transactions between employees	• Id, User • Date • pc • To, From • Cc, Bcc • Size • Attachment • content
http.csv	• Record the url visited by each employee	• id, user • Date • Pc • url • content
logon.csv	• Record the logon and logoff activities by each employee	• Id, user • date • Pc • activity

Figure 3.7: CERT Dataset

The data of the CERT data set can be classified into two categories:

1. **User behavior Data**: A user's behavior is characterized as a sequence of activities users perform in the organization. User's behavior data are from different real-time logging systems, such as login, device, file, email, and http logs. They are often required to be gathered and handled in a timely way to quickly detect and respond to anomaly behaviors. They are saved in http.csv, logon.csv, device.csv, file.csv, and email.csv.

2. **User Profile Data**: A user's profile information means the user's background or context data, which can be the employee's position, role, relationships with other users, permitted access, psychometrics, etc. They are saved in LDAP.csv and psychometric.csv.

There are multiple versions of the CERT datasets based on the generated time: The most commonly used version is r4.2 while the least used version is r6.2. CERT r4.2 mimics a corporation with 1000 employees and 32,770,227 activities, where 70 employees are malicious users and 7233 activities are anomalous in three scenarios of insider risk. The three scenarios are data exfiltration, embezzling intellectual property, and system sabotage. In contrast, CERT r6.2 illustrates a much larger corporation with 4000 employees and 135,117,169 activities, where only five users are malicious and only 470 activities are anomalous. There are five scenarios of insider risk, but only one malicious employee for each scenario. The newly added scenarios 4 and 5 are all about intellectual property theft, but they all occurred in one day.

- **Scenario 1**: The insider 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 thereafter.

- **Scenario 2**: The insider begins surfing job websites and soliciting employment from a competitor. Before leaving the company, they use a thumb drive (at markedly higher rates than their previous activity) to steal data.

- **Scenario 3**: A system administrator becomes disgruntled. Downloads a keylogger and uses a thumb drive to transfer it to his supervisor's machine. The next day, he uses the collected keyloggers to log in as his supervisor and send out an alarming mass email, causing panic in the organization. He leaves the organization immediately.

- **Scenario 4**: A user logs into another user's machine and searches for interesting files, emailing to their home email. This behavior occurs more and more frequently over a 3-month period.

- **Scenario 5**: A member of a group decimated by layoffs uploads documents to Dropbox, planning to use them for personal gain.

Compared with CERT r4.2, CERT r6.2 is a sporadic dataset, and thus insider threats are much more difficult to detect because it contains much fewer malicious users and their activities, but much more users' normal behaviors. We performed our experiments on both the CERT v4.2 and v6.2 datasets. To overcome the challenges of the CERT v6.2 dataset, we found good data augmentation approaches to generate synthetic data to overcome data imbalance and excellent deep learning models to model the users' behavior patterns instead of shallow machine learning models and manual feature engineering.

3.3.2 Data Engineering

In contrast to other related studies, this study uses the Natural Language Processing (NLP) approach to detect insider threats. This method considers each activity as a word, a user's daily activity sequence as a sentence, the entire set of users' activity sequences as a book, and user behavior patterns as hidden seminar rules. This study not only uses NPL to process data, encode and tokenize data, and perform an NPL model based on a self-attention mechanism to detect anomalous user behavior, but also uses the NPL method to perform data augmentation to overcome the data imbalance.

3.3.2.1 Downsampling Majority Class in CERT r4.2

Compared with CERT r6.2, CERT r4.2 is a dense dataset because it has many more malicious users and activities. However, this dataset is still imbalanced. We must down-sample the activity data of normal users to reduce the imbalance. To determine which down-sample algorithm is the best, this study applied RandomUnderSampler, AllKNN, NearMiss, and Tomek Links algorithms to generate downsized samples and then split them into the training dataset and the test dataset. The Gaussian NB classifier was trained on the training dataset. Next, we used the trained model to classify the test dataset and measure the prediction accuracy. The results are shown in 3.8, where Tomek Links performed the best; therefore, this study chose Tomek Links to down-sample normal users' behavior data.

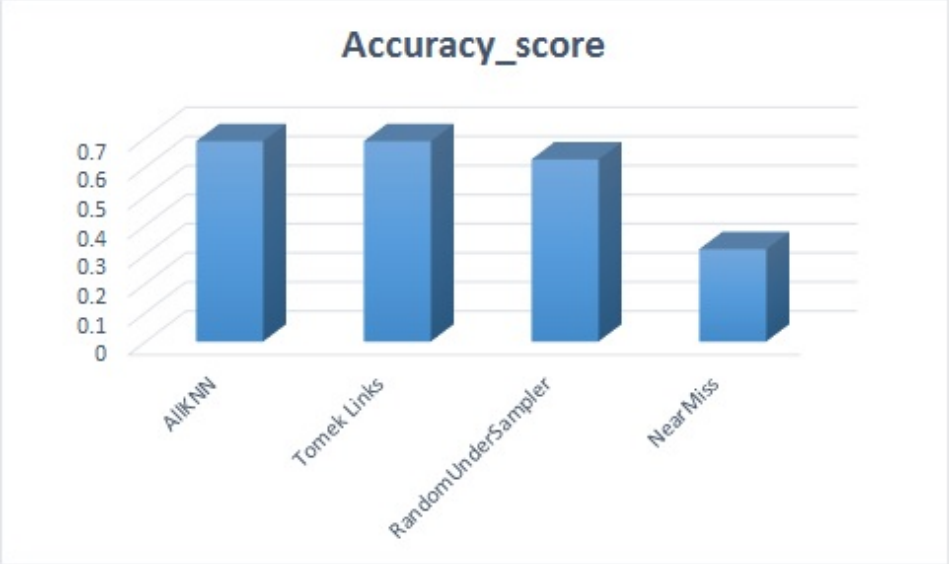

Figure 3.8: Accuracy Score of Downsampling Algorithms

3.3.2.2 Upsampling Minority class in CERT r6.2

Because there are very few anomalous user behavior data in the experiment dataset CERT r6.2, the dataset is extremely imbalanced. Whatever the models we use, they will always predict the class with the majority class because we do not have enough minority

class data to build robust models to distinguish classes. This study has to synthesize new data from the original dataset to up-sample the minority class and thus decrease the imbalance of the data.

Scholars have often used the Synthetic Minority Oversampling Technique (SMOTE) algorithm [58] to address this issue. The Smote algorithm is not a good fit for time-series data but fits scalar features. We can vectorize the text using the Tfidf Vectorizer and then apply the SMOTE algorithm. However, transformer-based models such as BERT and Roberta often have their own contextual tokenization. To optimize the performance of the models, this study adopts text augmentation approaches in the natural language process to upsample minority classes. However, augmenting text data is much more difficult than augmenting images (rotation, flip, sharpening, cropping, etc.). We have to ensure that synthetic data does not change the user's hidden behavior pattern (seminar and grammar) in user activity sequences (sentence). The simplest method is to insert or replace words using static word embeddings, such as GloVe or Word2vec. However, just as not every word has a synonym, not every activity can be replaced by another one. Even if you find one to replace it, the meaning of this time sequence can be totally changed because the context is different. To solve this issue, contextual word or sentence embedding methods are used to insert or replace target words, because they can consider surrounding words to generate different words in different contexts.

1. **Word Level Augmentation:** This chapter uses a powerful pre-trained transformer model, BERT, to insert or substitute activities (words) to generate synthetic data to upsample the minority class. The BERT model [8] is a pre-trained transformer-based model developed by Google. There are two models for the original English language BERT: the BERT-based model and the BERT-large model. We select a BERT-based model that has 12 encoders with approximately 110 million parameters. Instead of randomly selecting one word, this study uses the BERT model to predict possible inserted or substituted words based on the context it prelearned. Rather than inserting a word embedding or substituting a word with a

static synonym, the BERT model takes the surrounding words in the context as input to predict the target word and we then use the predicted target word to insert or substitute a word in the user activity sequence (sentence). Because the target word can occur in any position of the sentence, as a pre-trained transformer-based model for language understanding, BERT is good at learning both leftward and rightward contexts due to its bi-directional architecture.

2. **Sentence Level Augmentation** This study also generates a new synthetic user activity sequence (sentence) using a pre-trained transformer model GPT-2. Unlike the former method, we do not insert or replace a single or few activities (words), but instead generate the whole user activity sequence (sentence). This study used a pre-trained transformer-based model named GPT-2, which is well-known for language generation. GPT is a family of pre-trained varied transformer models for generative language learning, which was proposed by OpenAI in 2018. In contrast to large language understanding pre-trained models such as BERT or RoBERTa, the unique features of GPT [46] is its ability to generate coherent and contextually relevant text given a prompt or input because it is designed for autoregressive text generation tasks rather than sequence-to-sequence tasks like machine translation or text classification. GPT-1 is a 12-level transformer decoder with 12 heads, and a linear layer and a softmax layer are on top of that with 117 million parameters in total. GPT-2 is a scaling up of GPT-1 that shares the same architecture as GPT-1, but with modified normalization. Because we utilize the next sentence ability of GPT-2, the similarity of the generated synthetic data to the original data would be lower than that of the word level augmentation.

3.3.3 Feature Engineering and Semantic Vectorization

The feature engineering process is shown as follows:

1. The process starts by loading five domain data files (logon, device, file, email, and

HTTP) to their corresponding data subset after cleansing the data to remove dirty data and fill up missed values.

2. Extract and generate useful features from the data subsets: As shown in 3.9. this study extracts features such as user IDs, activity time, and activity types from existing data including Logon (Log On, Log off), Device(Connect, Disconnect), File(Copy File, Delete File, Open File, Write File), HTTP (WWW Download, WWW Upload, WWW Visit, and Email (Send Email, View Email), and created features such as Work Hours (In workhours, after work hours), PC (user's own PC, other user's PC), removable media (from removable media, to removable media, file type (exe file, non exe file), website (neutral website, job hunting, competitor, hacker, cloud storage), and email (send to internal, send to competitor, send to external non-competitor). We take 8:00 am to 5:00 pm as working hours and the period outside this range as the after-working hours. This study also regards the PC that a user uses most often as his/her assigned PC.

3. Encodes user behavior to a numeric token according to activity type, occurred time, and related features.

$$x = B * 24 + h \tag{3.1}$$

where x is a numeric token of the behavior that can distinctively identify user behavior, B is the behavior code based on activity type and related features (Figure 3.9), and h denotes the occurrence hour from the beginning of the day.

4. All data subsets are merged into one dataset. Group the data by each user and then by each day, and sort each user's behavior by the occurrence time to generate a user's behavior sequence for each day. Thus, a user's daily activities comprise a long string of activities on that day. For example, log on, visit the website, copy files, connect to the device, and send emails to colleagues. . . , log off, etc.

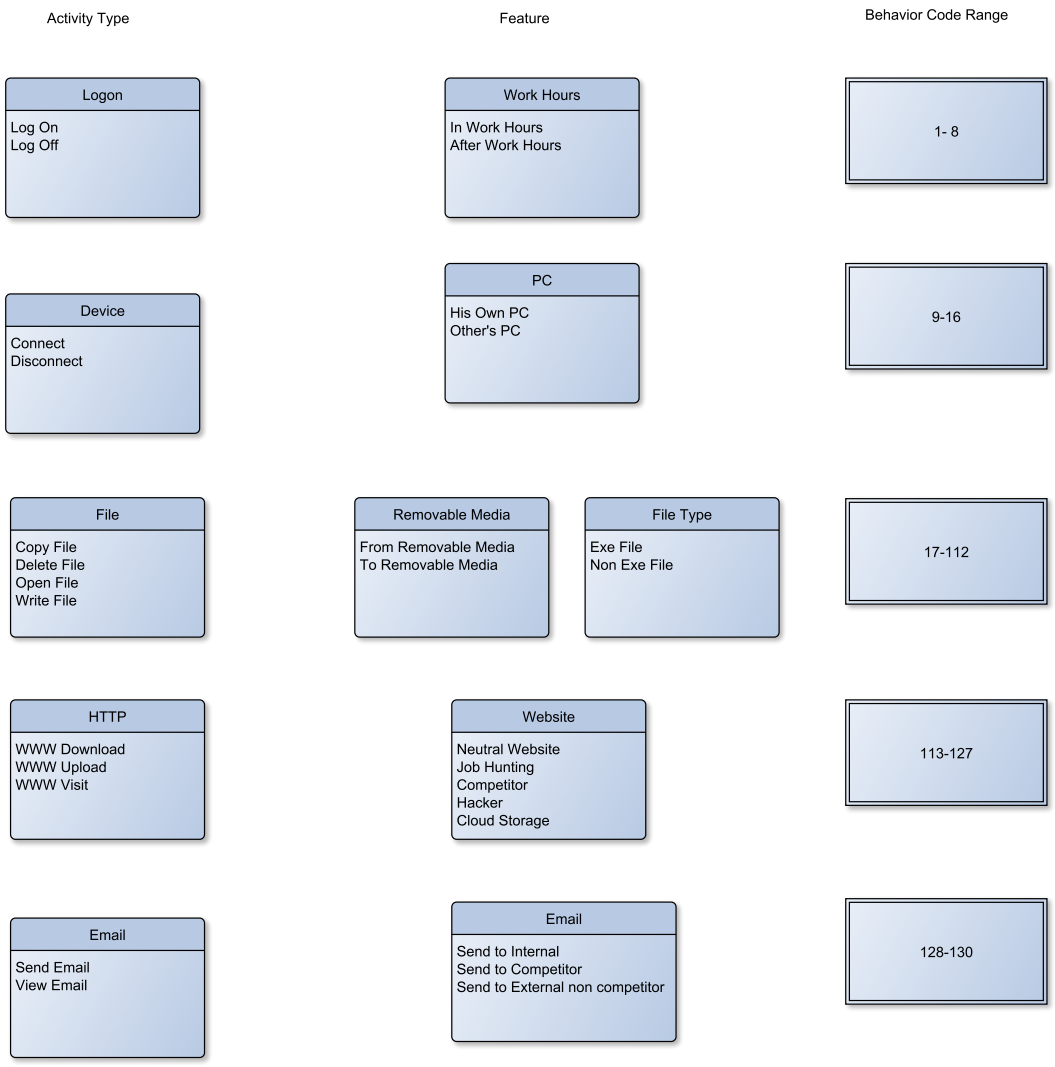

Figure 3.9: The Range of Behavior Code

5. Combine the dataset with insider.csv to label the user behavior sequence as normal or anomalous.

6. Text data are converted into numeric vectors. We accomplish this by transforming each word into a word embedding. This study takes the textual sequence of the user's daily activities as input obtains the vocabulary of the word index, transforms the input from a list of words into a list of indexes in the vocabulary, pads the sequence to have the same length using 0, and outputs a two-dimensional matrix whose every row is a dense vector containing the information of the word's meaning.

3.4 Detection Models

3.4.1 Transformer Model

The transformer model was introduced in 2017 by a team at Google Brain [44] and is increasingly becoming one of the first choices in Natural Language Processing (NLP). We applied this model to insider threat detection to capture the temporal dependency of user behavior. This model is the first sequence model that relies entirely on self-attention to model global temporal patterns between the input and output without any recurrent layers or convolutional layers that are most frequently used in temporal sequence modeling. Self-attention is an attention structure that associates different positions of a single sequence to output a depiction of the sequence. This study regards the input as a set of query matrix (Q) and key matrix (K) value matrix (V). Attention is calculated as a weighted sum of the values, where the weight corresponding to every value is the softmax function of the dot products of the query with keys, divided by the square root of d_k which helps prevent varnishing gradients in training.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.2}$$

Where d_k is the dimension of keys.

The original transformer has a typical encoder-decoder framework. The encoder converts an input sequence of symbol representations (x_1, ..., x_n) to a sequence of intermediate representations z = (z_1, ..., z_n). Then, the decoder converts z to an output sequence (y_1, ..., y_m). At every step, the model is autoregressive, which means that the output sequence depends linearly on its previous sequence and on a stochastic term.

The encoder consisted of a stack of six identical blocks. Each block has two sublayers. The first is a multi-head self-attention structure, while the second is a position-wise fully connected feed-forward network composed of two linear layers activated by the ReLu function. Similarly, the decoder consists of six blocks. The first difference from the encoder is that multi-head attention takes keys and values from the output of the encoder and queries from the output of the previous decoder layer. The second difference is that the first self-attention layer in the decoder block is masked to avoid attending to consequent positions, and the output embedding is right-shifted by one position to ensure that the predictions are autoregressive. The architecture of the original transformer is shown in Figure 3.10.

In contrast to RNNs, transformers simultaneously handle the entire input. The original input text is parsed into tokens by a byte pair encoding tokenizer, and the tokens are converted into equal-length vectors using word embedding and padding. The positional information of the token is then added to the word embedding.

$$\begin{aligned} PE(pos, 2i) &= sin(pos/10000^{2i/d_m}) \\ PE(pos, 2i+1) &= cos(pos/10000^{2i/d_m}) \end{aligned} \tag{3.3}$$

where PE is Positional Embedding, pos is the position, i is the dimension and d_m is the dimension of the input and output.

Next, encoders and decoders handle these vectors via multi-head self-attention to learn the attention weights of those representation subspaces and integrate such information from diverse positions. In particular, the masked multi-head self-attention sublayer is masked to avoid overfitting.

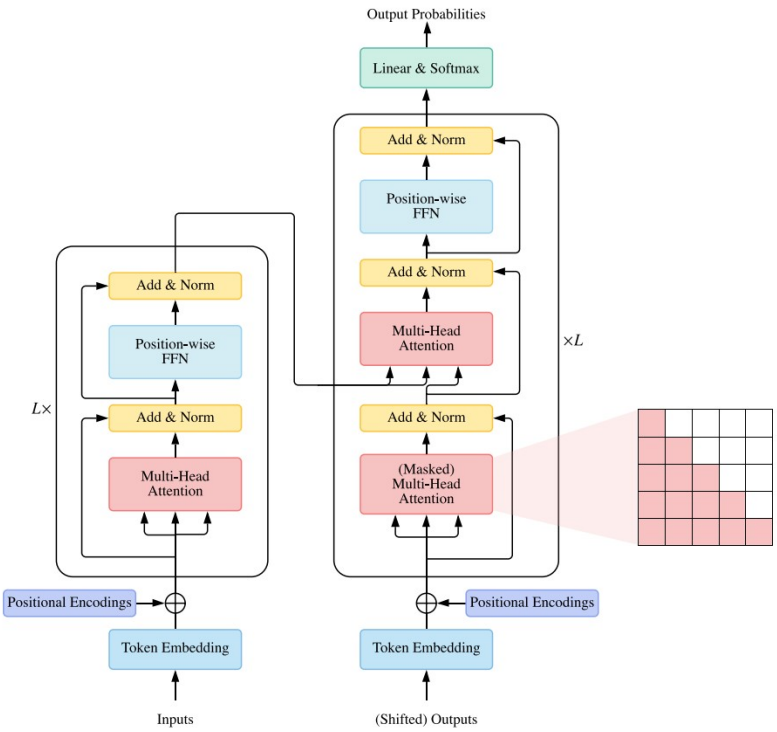

Figure 3.10: The Architecture of the Transformer Model

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{3.4}$$

$$Head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{3.5}$$

Where the parameter matrices $W_i^Q \in R^{d_m \times d_k}, W_i^K \in R^{d_m \times d_k}, W_i^v \in R^{d_m \times d_v}$, and $W_i^O \in R^{hd_v \times dm}$. A residual connection is built between the input's self-attention sublayer and the point-wise feed-forward network (FFN), preventing the vanishing gradient and the shift of the covariate. Subsequently, the computed results are refined by the normalization sublayer and point-wise feed-forward sublayer.

$$FFN(x) = \sigma(max(0, xW_1 + b_1)W_2 + b_2) \tag{3.6}$$

where FFN is the forward feed network, σ is the ReLU activation, W_1 and W_2 are the slopes, and b_1 and b_2 are the biases. Finally, the data go to a linear layer and a softmax

layer to obtain the malicious probability of user activities

Compared with CNN and RNN, including LSTM and GRU, the transformer utilizes self-attention to interactively compute a weighted sum of all input values to capture the global context of every word in the sequence. The complexity of the computation on each layer is much less. Additionally, the transformer has no recursive or convolutional mechanism, and the lengths of the paths traversing the network between positions in the input and output series are shorter. Thus, it is easier to learn long-term temporal patterns without the gradient disappearance. Moreover, RNN, including LSTM and GRU, must process in sequence so that they cannot utilize the parallel computing capability of the GPU. However, with multihead attention combined with positional embedding, the computation of the transformer can be processed in parallel to take all inputs simultaneously. Therefore, the transformer model has a much better training and inference efficiency and saves considerable training time. Furthermore, the transformer model makes fewer assumptions about the schema information of the data. Tokenization makes transformers a common and extensive architecture for processing heterogeneous input data.

3.4.2 Proposed Custom Model: DistilledTrans

To further improve the detection performance and reduce the training time, this study proposes a customized transformer model named DistilledTrans to detect insider threats. The architecture of the original transformer model is simplified to maintain only the encoder stack. As shown in Figure 3.11, a one-dimensional GlobalAveragePooling layer, dropout layer, and fully connected layer with sigmoid activation were added to the top of the encoder stacks to obtain the prediction results. In contrast to some scholars' conclusions, this study found that adding or reducing one encoder/decoder layer or putting a normalization layer before or after the multi-head self-attention mechanism or FFN achieved almost the same performance. However, adding too many blocks would degrade the detection performance. Therefore, the number of layers should match the size and complexity of the training dataset. Compared with other models, DistilledTrans

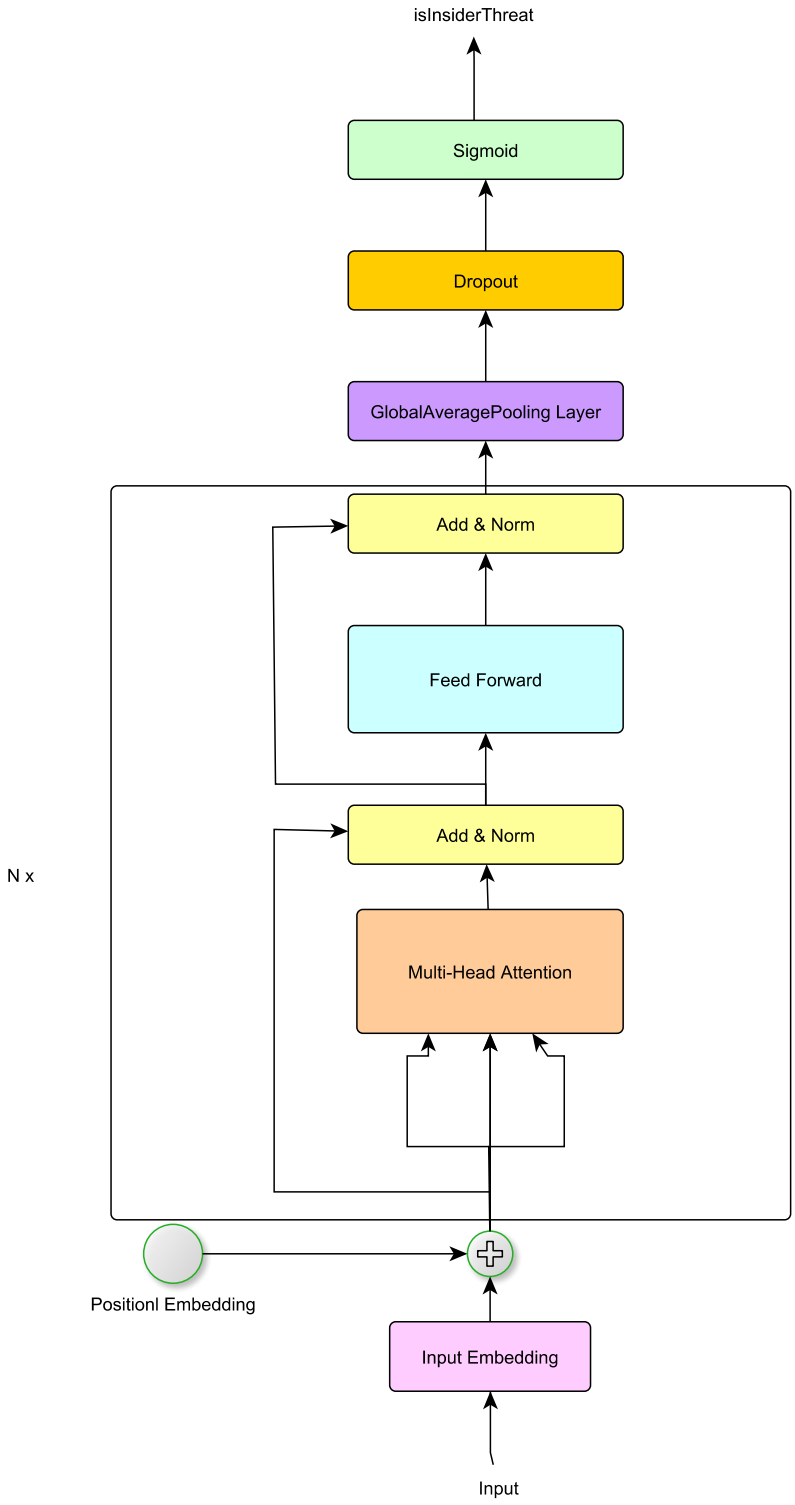

Figure 3.11: The Architecture of DistilledTrans Model

exhibited excellent detection performance across different datasets, including datasets augmented by different data augmentation approaches, sporadic datasets, and dense datasets.

3.4.3 Pre-trained Transformer Models

When building the transformer models, this study used an embedding layer to convert each integer index into a dense vector containing encoding. Because this embedding layer is task-independent, it cannot provide context-dependent word embedding. Thus, it may not be possible to learn sophisticated behavior patterns in a time series. In addition, the size of the CERT dataset may be insufficient to train a complicated corus. In contrast, pre-trained LLMs are trained on much larger datasets and have their own learned tokenizer. Therefore, to verify if there is room to further improve the detection performance, this study explored the full-tuning approach on pre-trained LLMs (BERT, RoBERTa, etc.) by adding the final layer to the models to detect insider threats. The final layer is composed of a dropout sublayer, a linear sublayer with ReLU activation, and a sigmoid sublayer. The architecture of the BERT model plus the final layer is shown in Figure 3.12

RoBERTa model plus the final layer has the same architecture as BERT model plus the final layer. This study also tuned the RoBERTa model to detect insider threats.

3.4.4 Hybrid Models

Hybrid models combining transformer-based models with other deep learning models have become an increasingly prevalent approach to accomplish various tasks in different domains. This study also built two hybrid models to perform insider threat detection: combining BERT with CNN and combining BERT with LSTM. In the BERT+CNN model, BERT is used as the low-level model and connected to the CNN with three convolution kernels, which acts as the high-level model. In the BERT+ LSTM model, BERT works as a low-level model and is connected to LSTM with two layers, which acts

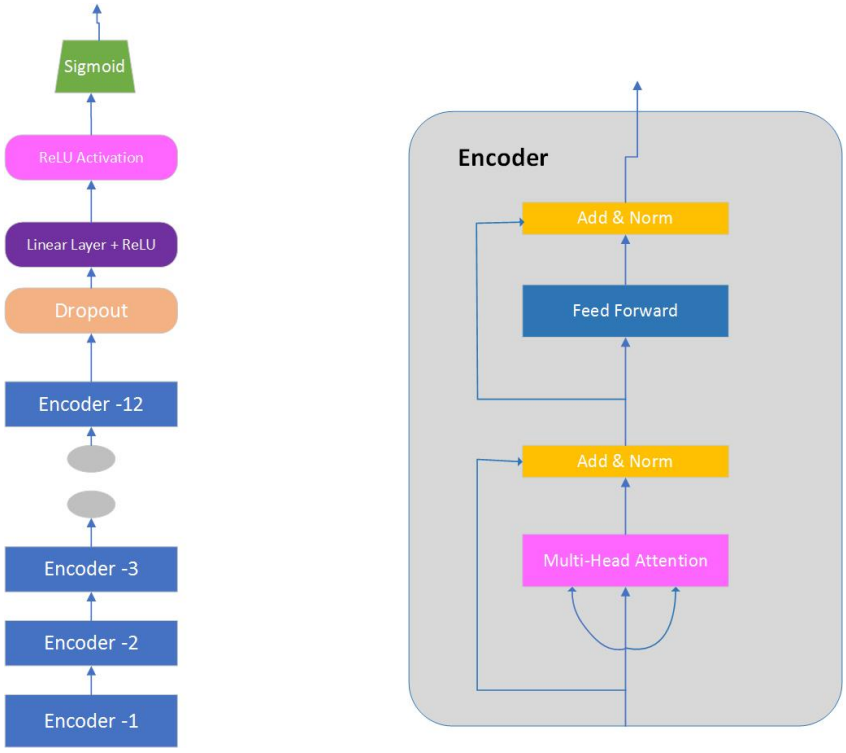

Figure 3.12: The Architecture of BERT Model Plus the Final Layer

as a high-level model. BERT is chosen as the low-level model because it showed better performance and higher stability in insider threat detection. On the other hand, CNN and LSTM are chosen as high-level models because they are among the most popular deep learning models applied in various learning tasks, especially insider threat detection. On the top of CNN or LSTM model, a dropout layer and a linear layer are added to generate the classification results. The architecture of BERT + LSTM model and RoBERTa + LSTM model are shown in Figure 3.13

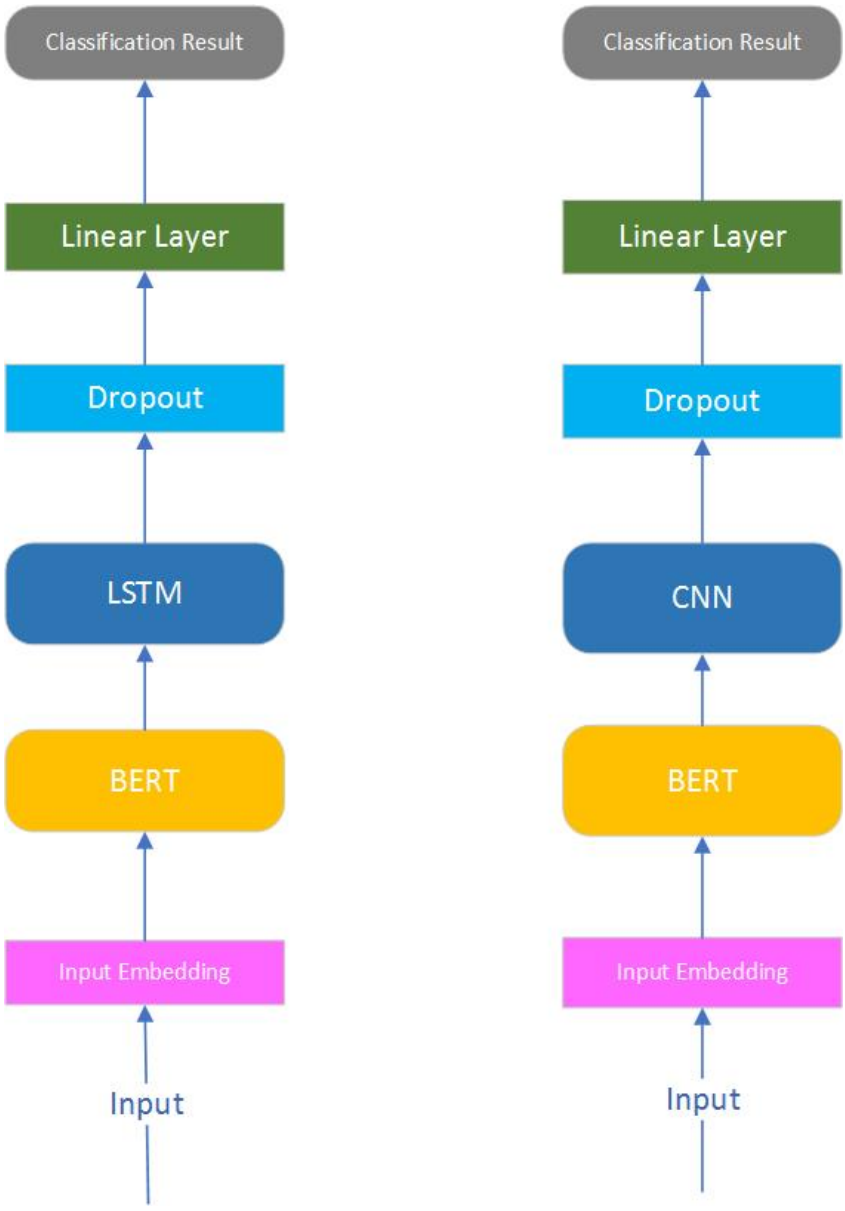

Figure 3.13: The Architecture of BERT + LSTM and BERT + CNN Model

3.5 Experiment

3.5.1 Implementation Details

3.5.1.1 Experiment Setting

Our experiments were accomplished in Google Colab, which is a Virtual Machine with NVIDIA A100-SXM4-40GB GPU, 83.48GB RAM, and 166.77 GB disk. The Pytorch version is 2.0.0, and the Python version is 3.10.11. The main software packages used to develop the models encompass pandas, Matplotlib, transformers, Pandas, Scikit-learn, and nlpaug. In this study, we trained our model with batch sizes of 32 and 20 epochs and a learning rate of 1e-6. The proposed models are implemented by the transformers package of Hugging face framework, e.g., BertModel class, RobertaModel class, etc.

This study trained the original transformer, DistilledTrans model, and fine-tuned BERT or RoBERTa based model on the CERT r4.2 dataset and the CERT r6.2 dataset augmented by contextual embedding words predicted by BERT model and contextual embedding sentence predicted by GPT-2 model. After data processing and augmentation, the ratio of normal behavior and abnormal behavior is 1:1 in the training datasets. We tested the models on the original CERT r4.2 dataset and CERT r6.2 dataset. Training Data was never used in the test.

3.5.1.2 Baselines

The CERT r4.2 dataset is a dense dataset containing more malicious use cases than the other versions, while CRET r6.2 has the least anomalous user behavior data. Most researchers worked on CERT r4.2. In this study, experiments were performed on both datasets. We adopted eleven baseline models on the CERT r4.2 dataset: One Class SVM [33], Hidden Markov model [59], Isolation Forest [60], and Decision Tree with undersampling approach [31] are typical applications of machine learning models for insider threat detection.The Deeplog [61], AutoEncoder, LSTM-AutoEncoder [62], LSTM-based

AutoEncoder [63], LSTM-CNN [63], and Hybrid Learning approach [40] are representative deep learning models for insider risks. Huang et al. [64] applied the RoBERTa + LSTM model, which is the only study that we found to use a transformer-based hybrid model to detect insider threats, but it only ran on CERT r4.2. Our evaluation baselines on the CERT r6.2 dataset includes classic machine learning algorithms, such as Isolation Forest and one-class SVM, and deep learning models, such as LSTM-RNN [65], multi-state LSTM + CNN [66], Hierarchical LSTMs [38], DeepMIT [67], log2vec,log2vec++ [68] using graphic embeddings, and DD-GCN[43]. However, most of these studies did not provide complete experimental results. Some only gave accuracy, some only gave AUC, and the rest only gave precision, recall, or F1 scores.

3.5.1.3 Evaluation Metrics

The main purpose of the experiments is to evaluate the performance of the proposed model in detecting insider's abnormal behavior. To evaluate our model, we used the following performance metrics as evaluation criteria:

- **Accuracy**

  Accuracy is the proportion of correctly classified instances in all instances. That means out of the total number of predictions made by the model, how many percentages we classify correctly,

  $$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.7}$$

  where TP, TN, FP, and FN refer to true positive, true negative, false positive, and false negative.

- **Recall**

  Recall is also called Detection Rate, Sensitivity, or True Positive Rate (TPR)

  $$Recall = \frac{TP}{TP + FN} \tag{3.8}$$

Recall is the proportion of positives correctly classified among the actual positives. That means out of the positive data, how many percentages we correctly diagnose as positive.

- **Precision**

  Precision is also called Confidence

$$Precision = \frac{TP}{TP + FP} \tag{3.9}$$

  Precision is the proportion of actual positives among the predicted positives. That means out of the diagnosed positive data, how many percentages do we classify correctly.

- **F1 Score**

  The F1 Score, also known as the F-score or F-measure, is the harmonic mean of precision and sensitivity. F-score is defined as the following:

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3.10}$$

  where precision is calculated as $\frac{TP}{TP+FP}$ and recall is defined as $\frac{TP}{TP+FN}$. Increase precision by trading off recall and vice versa. F1 score balances precision and recall, allowing for a trade-off between precision and recall.

- **Area under ROC Curve (AUC)**

  The Receiver Operating Characteristic (ROC) curve, is derived from the coverage curve through axis normalization to the range [0, 1]. It illustrates the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) as the discrimination threshold varies. The area under the ROC curve (AUC) quantifies the ranking accuracy of the model.

3.5.2 Evaluation and Analysis

3.5.2.1 Evaluation Results

On the dataset CERT r4.2, Table 3.1 shows that BERT + Final layer and RoBERTa + Final layer prove that they are reliable models that outperform all baselines in terms of almost all evaluation metrics except that the precision is slightly lower than that of LSTM-AutoEncoder and Isolation Forest. The original transformer demonstrated excellent performance with an AUC of 97.37%, an accuracy of 93.53%, and a the precision of 98.13%, outperforming all the baseline models. However, its recall is 80.71%, and thus F1-score is only 88.61% which is lower than that of RoBERTa-LSTM and LSTM-AutoEncoder. In comparison, DistilledTrans not only outperforms all baselines in terms of accuracy, precision, and AUC, but also increases recall to 84.62%; thus, the F1 score increases to 90.53%. However, the recall and F1 score still lag behind slightly with the LSTM-AutoEncoder, although its training time is more than 10 times faster.

As for the CERT r6.2 augmented by contextual word embedding, Table 3.2 shows that DistilledTrans outperforms other transformer-based models and all baselines in terms of almost all evaluation metrics except that the recall is slightly lower than LSTM-RNN and DeepMIT. The original transformer, BERT + FL, and BERT + CNN also outperform all the baselines in terms of AUC and accuracy. Additionally, BERT+FL and BERT+ CNN performed the best in the recall, but their precision and F1 scores need improvement.

As for the CERT r6.2 augmented by contextual embedding sentences, Table 3.3 shows that DistilledTrans performs the best among all models. The original transformer ranked second. Their metric values are so excellent that they outperform other transformer-based models and all baselines in all metrics, including AUC, F1-score, recall, precision, accuracy, and training time. We combine CNN or LSTM with BERT to train and test again; they showed some improvement, but their performance, especially recall, F1 score, and AUC, still could not compete with DistilledTrans and Original Transformer.

Table 3.1: Performance comparison of the models trained by CERT r4.2

Algorithms	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**
Original Transformer	**93.53%**	**98.13%**	80.77%	88.61%	**97.37%**
DistilledTrans	**94.48%**	**97.35%**	84.62%	90.53%	**97.18%**
BERT + FL	**96.40%**	93.23%	**95.38%**	**96.40%**	**96.12%**
Roberta+ FL	**96.64%**	94.62%	**94.62%**	**94.62%**	**96.09%**
ITDBERT: Roberta + LSTM	X	93.00%	91.87%	92.43%	95.56%
LSTM - CNN	X	X	X	X	94.49%
Deeplog	X	72.59%	90.26%	80.47%	X
AutoEncoder	X	50.18%	94.10%	65.46%	X
LSTM based Autoencoder	90.00%	X	X	X	X
LSTM-AutoEncoder	90.00%	97.00%	X	94.00%	X
Hidden Markov Model	X	X	X	X	83.00%
Isolation Forest	79.00%	95.00%	35.19%	51.35%	X
One class SVM	87.79%	X	X	X	X
DT + Under-sampling	X	X	88.00%	X	94.00%
Hybrid Learning Approach	88.00%	85.70%	84.40%	84.85%	84.00%

Note: The bold Digital indicates that the metric's value is higher than all baselines' values. X indicates that the performance metrics are not provided.

Table 3.2: Performance Comparison of the Models Trained by CERT r6.2 Data Augmented by Contextual Word Embedding

Algorithms	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**
Original Transformer	**99.35%**	**100.00%**	76.67%	86.79%	**99.50%**
DistilledTrans	**99.72%**	**100.00%**	90.00%	**94.74%**	**97.52%**
BERT+ FL	**99.35%**	84.85%	**93.33%**	88.89%	**96.42%**
BERT + CNN	**99.08%**	77.78%	**93.33%**	84.85%	**96.29%**
BERT + LSTM	96.58%	44.26%	90.00%	59.34%	93.38%
LSTM-RNN	93.85%	95.12%	92.46%	X	X
multistate LSTM + CNN	85.00%	X	X	X	90.47%
Iforest	X	X	X	X	76.19%
OCSVM	X	X	X	X	73.67%
Hierarchical LSTMs	X	X	X	X	94.96%
DeepMIT	X	91.60%	93.20%	93.20%	X
log2vec	X	X	X	X	86.00%
log2vec++	X	X	X	X	93%
LSTM	X	X	X	X	71.00%
DD-GCN	98.65%	X	X	85.69%	X

Note: The bold Digital indicates that the metric's value is higher than all baselines' values. X indicates that the performance metrics are not provided.

Table 3.3: Performance comparison of the models trained by cert r6.2 augmented by contextual embedding sentence

Algorithms	**Accuracy**	**Precision**	**Recall**	**F1-Score**	**AUC**
Original Transformer	**99.82%**	**100.00%**	**93.33%**	**96.55%**	**98.52%**
DistilledTrans	**99.82%**	**100.00%**	**93.33%**	**96.55%**	**99.63%**
BERT + CNN	97.97%	**100.00%**	26.67%	42.11%	63.33%
BERT + LSTM	**98.80%**	**100.00%**	56.67%	72.34%	78.33%
LSTM-RNN	93.85%	95.12%	92.46%	X	X
multistate LSTM + CNN	85.00%	X	X	X	90.47%
Iforest	X	X	X	X	76.19%
OCSVM	X	X	X	X	73.67%
Hierarchical LSTMs	X	X	X	X	94.96%
DeepMIT	X	91.60%	93.20%	93.20%	X
log2vec	X	X	X	X	86.00%
log2vec++	X	X	X	X	93%
LSTM	X	X	X	X	71.00%
DD-GCN	98.65%	X	X	85.69%	X

3.5.2.2 Comparison Analysis

From table 3.1, 3.2, and 3.3, we can see that whether trained on dense dataset CERT r4.2, or sporadic dataset CERT r6.2 augmented by contextual word embedding and contextual sentence, both the original Transformer and DistilledTrans performed very well. They take the lead of other models in almost all performance metrics, meanwhile, their structure is more concise, and the training time is much shorter. The self-attention mechanism, positional embedding, and multiple heads enable them to have a strong ability to learn long temporal relationships, compute in parallel, and easily integrate data without knowing the data schema. Contextual independent word embedding results in excellent extensivity when trained on different datasets, including augmented data sets. Compared with the original transformer, DistilledTrans performs better because it only utilizes encoder blocks, and thus avoids the over-processing of information. This also demonstrates that our data augmentation methods are feasible for training an excellent model for detecting insider threats.

For the dense dataset CERT r4.2, fine-tuning the pre-trained BERT or RoBERTa plus a final layer can further improve recall with a slight sacrifice of precision. Because they are pre-trained on large datasets, we can take advantage of the pre-trained knowledge to perform the downstream task of insider threat detection.

As for the sporadic dataset, CERT r6.2 augmented by contextual word embedding predicted by the BERT model, BERT plus a final layer can further improve recall with a slight decrease in precision owing to pre-trained word embedding and model structure. In contrast, the performance of BERT + CNN and BERT+ LSTM is worse, especially in terms of precision and F1 score. This shows that the self-attention structure is a substitution for convolutional and recursive mechanisms rather than a supplement. Putting another neural network on BERT makes the structure too deep and complex, so information would be lost, thus damaging the performance. Therefore, a hybrid model is not a suitable choice for insider threat detection.

For the sparse dataset CERT r6.2, augmented by contextual embedding sentences pre-

dicted by the GPT-2 model, both the original transformer and DistilledTrans perform very well. In contrast, neither BERT + CNN nor BERT+ LSTM perform satisfactorily. Their recalls are much worse than their performance in CERT r6.2 augmented by the contextual embedding word method predicted by the BERT model. BERT plus a final layer cannot detect insider threats when the dataset is augmented by another model, GPT-2. Because BERT's contextual embedding is no longer effective. Even if the neural network is added to the BERT model, the performance of the model cannot still match that of two transformers (original transformer and DistilledTrans). In addition, RoBERTa plus a final layer cannot detect insider risks after training on the augmented CERT r6.2 dataset using the contextual embedding word method predicted by the BERT model or the contextual embedding sentence method predicted by the GPT-2 model. This is due to a mismatch between its contextual embeddings and the embeddings of the augmented dataset predicted by other models. For instance, RoBERTa uses a large byte-level BPE vocabulary to build its own contextual-dependent tokens, so it may not be able to understand our input embedding predicted by the BERT model or the GPT-2 model at all.

The reasons that DistilledTrans performs the best are as follows:

1. First, it employs self-attention in a simplified architecture to effectively capture the global context of each action in a sequence, making it well-suited for understanding complex, hidden, and non-linear user behaviors with contextual information and accurately identifying subtle anomalies that are small deviations from normal user behavior profiles

2. Second, Shorter paths between input and output positions avoid overprocessing information and facilitate learning long-term temporal patterns

3. Third, it requires fewer assumptions about the data's schema information; independent tokenization make it easy to seamlessly process heterogeneous input data especially augmented data generated by different transformer variant models

Chapter 4

FedITD: Insider Threat Detection Framework Based on Federated Learning

In this chapter, we present and evaluate our proposed Federated Parameter-Efficient Tuning with Pre-trained Language Models and Transfer Learning Framework for Insider Threat Detection - FedITD. We first provide an overview of the proposed system in section 4.1. The details of the proposed framework are then introduced in section 4.2. The evaluation and analysis for the proposed FedITD is shown in section 4.3.

4.1 Introduction

Addressing the existing problems that insider threat detection faces requires a combination of multiple advanced technologies: We adopt transformer variant models to improve poor detection performance and accurately identify anomalies. To address the inefficient model, we use PETuning methods to tune models. To break the barriers between isolated data islands and detect unknown attacks, we utilize federated learning and transfer learning to learn from other clients. To overcome highly imbalanced dataset issues, Natural Language Processing (NLP) data augmentation methods are applied to upsample minorities. To handle lacking personalized models and labeled data, we incorporate transfer

learning unsupervised domain adaption. To solve long latency and high resource costs, we design a flexible, hierarchical, and federated framework with local real-time detection systems, locally trained models, and federated PETuning methods. Therefore, We propose a novel system called FedITD combining pre-trained Large Language Models, PETuning, federated learning, and transfer learning.

To overcome this obstacle of data integration, FL has surfaced as a privacy-conscious technique. FL aims to collectively train models without the need to transmit substantial client data to a centralized location. In FL, decentralized clients only simply compute and send a small part of model information, like parameters or gradients, to a server at intervals, where they are then aggregated to generate a global model. It disrupts data islands by enabling learning across distributed data sources. It accelerates training and reduces communication requirements compared to centralized learning methods. Additionally, it facilitates collaboration without compromising the security of each participant by maintaining data locally. Thus, FL is chosen as our system's foundation.

Pre-trained LLMs like BERT [8], RoBERTa [9], XLNet [10], and DistllBERT[11] have showcased remarkable performance across various natural language processing (NLP) tasks, such as GLUE [69]. Wang et al. [12] demonstrated exceptional insider threat detection performance by centrally fine full-tuning the pre-trained LLMs on the CERT dataset. Therefore, we adopt Pre-trained LLMs as the backbone models of our FL system. However, LLMs introduce challenges such as significant communication overhead, storage cost, memory usage, high costs associated with adapting local models after FL, and security concerns.

PETuning methods, including Adapter [13], LoRA [14], and BitFit[15], typically involve locking most of the parameters in PLMs and only tuning a small subset of extra parameters or a small portion of the initial model parameters for downstream jobs. These methods were originally trained centrally. We introduce them to FL to effectively reduce communication costs and resource usage while maintaining satisfactory performance. It also can significantly improve security e.g., counter gradient inversion attacks, which can

result in an average reduction of 40.7% in the accuracy of recovered words compared to Federated Parameter Full Tuning [16].

However, the global model may not be a good fit for a specific client due to local data heterogeneity and diversity. Using Transfer Learning (TL), this study can rapidly develop a custom local model tailored to specific organizations or cases, leveraging insights gained from previous experiences shared by other organizations. The central concept is to minimize the distribution divergence between various organizations. It significantly enhances adaptability because this approach could offer multiple variations of the same global models customized for different organizations or use cases with their limited local data, thus yielding superior local results. We combine federated PETuing and TL to address the aforementioned challenges and further improve unknown attack detection.

4.2 Proposed Framework

4.2.1 Problem Statement

Insider threat detection is a critical challenge, as malicious activities from internal users often evade traditional security systems. Centralized machine learning approaches are effective but raise a variety of challenges, especially concerns about data scarcity and privacy. The primary goal of this thesis is to address these issues by utilizing federated learning in conjunction with transfer learning techniques to collaboratively train a model for insider threat detection without the need for direct data exchange, maintaining data privacy, and solving data scarcity while achieving similar performance to a centralized model. We have data from K different clients, i.e., organizations, which is indicated by $\{C_1, C_2, \cdot, C_K\}$ and the data collected by each client is symbolized by $\{D_1, D_2, \cdot, D_K\}$. Traditional methods typically train a unified model M_u by aggregating all the clients' data, $D = D_1 \cup D_2 \cup \ldots \cup D_K$. However, this method has to share all the clients' data and thus compromises clients' privacy. Denote a customized client model locally trained based only on the local dataset D_k, k $\in [1, K]$ as M_c

In our problem, our objective is to collaboratively use all clients' information to train a federated transfer learning model M_f, while ensuring each client $C_k, k \in [1, K]$ does not disclose its data D_k and keep their privacy. The performance of M_c, M_u, and M_f are symbolized as P_c, P_u, and P_f respectively. Additionally, another goal is to build an effective framework to enhance P_f to be better than P_c and as close as possible to P_u. Moreover, this framework should be highly efficient in tuning Pre-trained LLMs so we can save training time, communication cost, and resource consumption. Furthermore, M_f is required to be highly adaptable to the C_k's local data distribution and possesses the ability to generalize well to detect unknown attacks.

$$\frac{|P_u - P_f|}{P_u} < \Delta \tag{4.1}$$

where Δ is a very small nonnegative percentage, e.g., 6%.

4.2.2 Framework Overview

As Figure 4.1 shows, the FedITD framework is composed of three systems: the supervised system, the client security system, and the federal system.

1. **The supervised system** encompasses various aspects of the monitored system, such as PCs, file systems, emails, web browsing, devices, etc. The collected data can pertain to either user behavior or user profile. User behavior information is derived from various real-time logging systems, including host logs, network logs, VPNs, firewalls, etc. User profile data, on the other hand, refers to users' background details or contextual information such as user identities, positions, relationships, permitted access, and psychometric attributes. It's worth noting that different clients' supervised systems are separated at various locations and operate independently without sharing information.

2. The middle layer is **the client security system** serving as the core of the framework. It is specifically designed to accommodate insider threat detection constituents and act as local access points to federal systems. Consequently, each

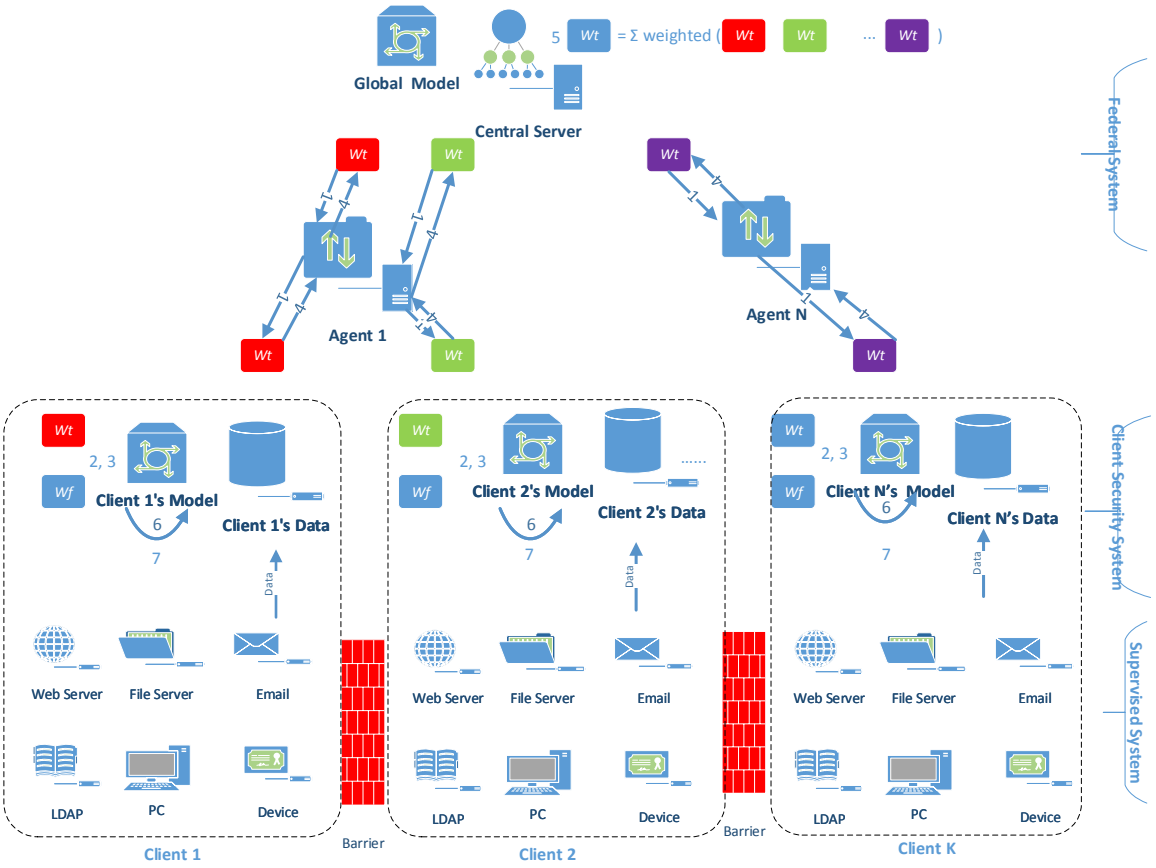

Figure 4.1: FedITD Framework and the Learning Process

client security system is empowered to gather data from the respective supervised systems for training the client models and detecting potential attacks. This layer encompasses the entire workflow of insider threat detection, including data processing, model training, pre-trained model tuning, detection, analysis, mitigation, and alert generation. It is assumed that this subsystem possesses sufficient computing power and network bandwidth to preprocess data, execute federated learning, perform transfer learning, and conduct real-time anomaly detection using deep learning models. Meanwhile, it can centrally train its client model based on locally collected data.

3. **The federal system** is situated on the top and facilitates collaborating federated learning, particularly in terms of parameter aggregation and distribution. This study employs a centralized architecture within this system. This architecture comprises a central server located either on the cloud or on-premise, responsible for aggregating updated parameters from agents and training the global model. Additionally, it encompasses agents, which act as sub-servers facilitating the transmission of updated parameters from clients or optional training segmented models tailored to client groups based on their performance or characteristics. These agents also serve as access gateways to the central server for clients.

This system can also be implemented using a **decentralized architecture**. This involves establishing an agent peer-to-peer network, wherein clients collaboratively train deep learning models with high performance, using consensus and incentive mechanisms embedded in the blockchain [70].

Security Interface: Homomorphic encryption is employed to aggregate the update parameters without the server knowing the generated model. The Paillier cryptosystem [71] supports addition and can disseminate the resulting aggregate back to the clients. Subsequently, each client can decrypt the updated parameters by dividing biases and weights by the number of participants to obtain the averaged values.

The main learning process is as follows:

1. Initially, the global model residing on the central server undergoes training using public datasets or existing collected data. The initial model is then distributed to all clients.

2. Subsequently, clients assemble the global model using downloaded lightweight trainable parameters W_t^0 and the backbone Pre-trained LLM's frozen parameters $W_f^{k,0}$, k $\in [1, K]$. Perform transfer learning to overcome the difference in data distribution between server data and client data.

3. Following this, clients train the assembled model with their respective private local data D_k. PETuning methods are employed to streamline parameter communication, exchanging only lightweight trainable parameters instead of all the cumbersome Pre-trained LLM parameters.

4. Post local training, clients transmit their updated efficient parameters, e.g., the k-th client's $W_t^{k,1}$, to the central server for federated aggregation.

5. The server conducts federated aggregation based on the gathered parameters from clients and sends the updated parameter values, e.g., W_t^2 back to the clients.

6. Each client trains their client model again on their local dataset for a certain number of local epochs to get a better client model.

   Repeat 4), 5), and 6) until convergence of the model or the maximum number of global training rounds R is reached.

7. Clients Perform transfer learning to overcome the difference in data distribution between server data and client data

Importantly, it's worth noting that FL serves the purpose of knowledge sharing among all agents through model parameter aggregation. All parameter-sharing processes are conducted without risks of user data or sensitive information leakage. This is achieved

through techniques such as homomorphic encryption, Parameter Efficient Tuning, local model updates, federated averaging, and Federated Transfer Learning, ensuring the confidentiality and privacy of user data throughout the entire system.

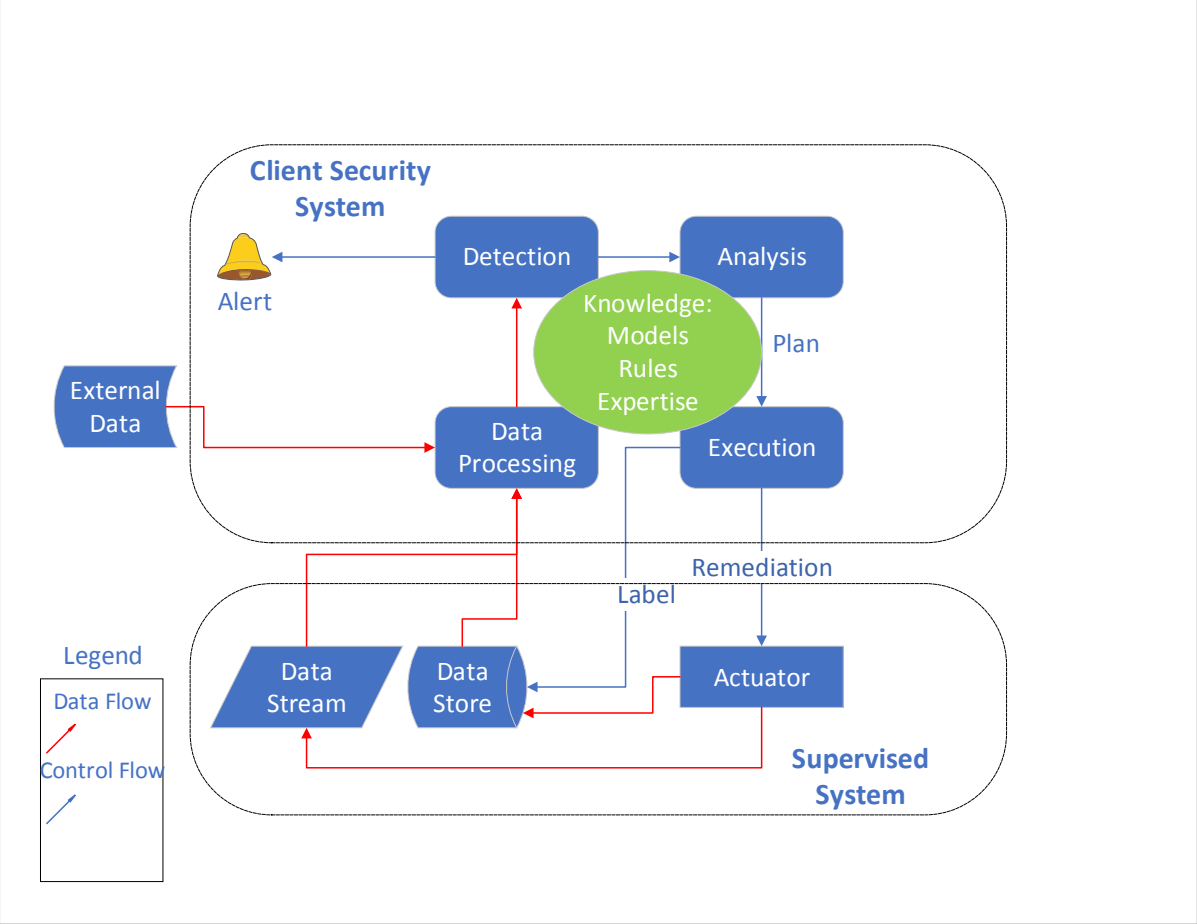

Figure 4.2: Insider Threat Detection Workflow

Figure 4.2 illustrates the workflow of insider threat detection running in the FedITD framework. Data stream/data store, data processing, detection, analysis, execution, actuator, and knowledge form a complete feedback loop. Knowledge is shared among all steps and plays a central role in this workflow. Deep learning models, defined rules, and experts' expertise play the role of knowledge. The client security system's data input is from the supervised system and external data sources (open dataset, shared data from partners, etc.). The supervised system collects data from either real-time data streams from sensors and devices or data stores (database, data warehouse, distributed

file system, etc.) persisting various hosts, devices, network, and system logs. Collected data are preprocessed in data processing and then fed into the rule engine or trigger the deep learning model inference pipeline running fine-tuned models for detection. If there is an abnormal user or activity, the security system will create an incident, raise an alert, and send all related information to the security analyst for analysis. Next, after in-depth analysis, the security analyst can label it as normal behavior or launch an automatic remediation workflow to take action. For instance, he can block the user so that it does not have access to the system or revert the damages done by the user's malicious actions. The actuator executes remediation actions and thus generates new data. So the cycle repeats.

Insider threat mitigation strategies often require detailed investigation, correlation with contextual data, and automated responses. They may include:

- **Containment**: Take immediate steps to contain the threat, such as revoking access, isolating affected systems, or restricting network connectivity. Implement both short-term measures to stop ongoing malicious activities and long-term strategies to prevent recurrence

- **Eradication**: Perform root cause analysis to investigate the incident to determine the root cause and eliminate it. This might involve removing malware, cleaning affected systems, closing vulnerabilities, or addressing policy violations.

- **Recovery**: Restore systems to normal operation, ensuring they are free from vulnerabilities or threats. Increase monitoring of the environment to detect any signs of remaining threats or new attacks.

- **Lessons Learned**: Conduct a thorough review of the incident, including what happened, how it was handled, and what could be improved. Document all findings, actions taken, and lessons learned to enhance future insider risk management. Update policies, procedures, and training programs based on the insights gained from the incident.

4.2.3 Pre-trained Language Models

This study selects the following pre-trained language models as the foundation models:

4.2.3.1 BERT

Bidirectional Encoder Representations from Transformers(BERT) is a pre-trained language model developed by Google [8] in 2018. It has significantly advanced the field of natural language processing (NLP) since its introduction in 2018. BERT is designed to understand the context of a word in search queries and other texts. Unlike previous models that read text input sequentially (left-to-right or right-to-left), BERT reads the entire sequence of words at once using a bidirectional self-attention mechanism, allowing it to capture the full context of a word by looking at the words from both sides of a word, which is crucial for understanding nuanced meanings in sentences. BERT is pre-trained on a massive corpus of text data, such as Wikipedia (2,500M words) and BooksCorpus (800M words). During this pre-training phase, BERT randomly masks some words in the input sentence and tries to predict them based on the context provided by the unmasked words. This task helps the model learn contextual relationships between words. BERT is also pre-trained to predict whether a given sentence B is likely to follow sentence A to learn relationships between sentences. This task is beneficial for understanding the coherence and context across multiple sentences. After pre-training, BERT is more flexible and can be fine-tuned to adapt its general language understanding to the requirements of a wide range of NLP tasks with minimal modifications, for example, question answering, text classification, Named Entity Recognition (NER), Text Summarization, translation, etc.

BERT models only have the encoder part of the original transformer model but are generally deeper and more complex than the transformer model. There are two models for the original BERT model:

1. BERT-based model:12 encoders with approximately 110 million parameters. There

are 12 bidirectional self-attention heads and 768 hidden layers in every block.

2. BERT-large model: 24 encoders with approximately 340 million parameters. There are 24 bidirectional self-attention heads and 768 hidden layers in every block

The bidirectional nature of BERT allows for a better understanding of context, leading to more accurate interpretations of words and sentences. Additionally, its ability to be fine-tuned for different tasks makes it a versatile tool for many NLP applications. Its introduction has led to significant improvements in the performance of many NLP applications, outperforming previous state-of-the-art models. Therefore, BERT has revolutionized NLP by setting new benchmarks on various NLP tasks, making it a baseline model in the field of NLP. However, the BERT model has limitations: by using masks to obscure parts of the input, the BERT model fails to account for the interdependence of the masked positions, leading to a discrepancy between the pre-trained model and the fine-tuned model. In addition, the BERT model is too large, slow, and heavy to fit resources constrained environment or real-time processing applications

This thesis adopts the BERT model as one of the foundation models to detect insider threats and also uses it to predict word embedding for insert and substitution based on the context it pre-learned to augment the minority class in the experiment dataset.

4.2.3.2 RoBERTa

Robustly Optimized BERT Approach(RoBERTa) is an advanced version of the model developed by Facebook AI [9] in 2019. RoBERTa retains the same transformer architecture as BERT, consisting of multiple layers of bidirectional transformers. However, RoBERTa improves on BERT by making several modifications to its training methodology to enhance performance on various NLP tasks:

1. **Training Data and Duration:** RoBERTa is trained on significantly more data compared to BERT. It uses a dataset that is ten times larger than BERT's (160GB), encompassing a diverse range of text sources like the Common Crawl News, OPEN-

WEBTEXT, and STORIES datasets. The model is trained for a longer duration, which helps in better convergence and understanding of language nuances.

2. **Removal of Next Sentence Prediction (NSP)**: RoBERTa removes the Next Sentence Prediction (NSP) task during pre-training, as experiments showed that eliminating NSP leads to improved performance.

3. **Dynamic Masking:** BERT uses a static masking approach where the same tokens are masked during each epoch of training. RoBERTa employs dynamic masking, where the tokens are chosen to be masked change during training. This results in more robust learning.

4. **Larger Batch Sizes and Learning Rates:** RoBERTa uses larger batch sizes and learning rates during training. This allows the model to better generalize and achieve higher accuracy on downstream tasks.

5. **Longer Sequences:** RoBERTa is trained with longer sequences of text, which helps in capturing more context and improving the model's understanding of the language.

6. **Tokenizer:** it adopts a byte-level Byte Paired Encoding (BPE), whose vocabulary is 50K sub-word units, as a tokenizer to replace the character-level BPE, whose vocabulary is 30K sub-word units.

RoBERTa has achieved state-of-the-art results on several benchmark NLP tasks, including General Language Understanding Evaluation(GLUE), Stanford Question Answering Dataset(SQuAD), and Reading Comprehension Dataset (SQuAD). Like BERT, RoBERTa can be fine-tuned for a wide range of NLP tasks, such as text classification, NER, question answering, sentiment analysis, text summarization, etc.

By refining the pre-training methodology and utilizing more extensive datasets, RoBERTa significantly outperforms BERT on various NLP tasks, demonstrating its robustness and

effectiveness. This study also selects the RoBERTa model as one of the foundation models to detect insider threats.

4.2.3.3 XLNet

eXtreme Language understanding NETwork(XLNet) is an advanced language model developed by researchers at Google and Carnegie Mellon University [10] in 2019. By relying on corrupting the input with masks, BERT overlooks the reliance between the masked positions, resulting in a divergence between the pre-trained model and the fine-tuned model. XLNet aims to overcome this limitation by employing a more sophisticated training approach and autoregressive method:

1. **Permutated Language Modeling:** Unlike BERT, which uses a masked language modeling (MLM) approach where some tokens are masked and predicted, XLNet employs a permutated language modeling objective. In this approach, the model learns all possible permutations of the sequence order rather than a fixed order to predict a token, which enables it to learn bidirectional contexts without explicitly masking any tokens. This is achieved by training the model to predict the probability of a token given all other tokens in the input sequence, irrespective of their position.

2. **Autoregressive Model:** XLNet is fundamentally an autoregressive model, meaning it predicts the next token in a sequence based on previous tokens. By combining this autoregressive nature with permutated language modeling, XLNet effectively captures bidirectional contexts while maintaining the benefits of autoregressive training.

3. **Transformer-XL Integration:** XLNet integrates Transformer-XL, a model known for handling long-term dependencies and sequences more efficiently, into pre-training to get long-term reliance in the input sequences. This is accomplished through a segment-level recurrence mechanism, enabling the model to preserve the memory

of the previous segment while handling the current segment. It allows the model to maintain context over longer sequences, which is beneficial for understanding lengthy texts.

4. **Training with Larger Data:** XLNet is trained on a larger and more diverse dataset compared to BERT. It utilizes over 130GB of text data from sources such as BooksCorpus, Wikipedia, Giga5, ClueWeb 2012-B, and Common Crawl.

XLNet retains the same transformer architecture as BERT but modifies the training process to accommodate the permutated language modeling objective and autoregressive nature.XLNet achieves state-of-the-art results on various NLP benchmarks by addressing the limitations of BERT's masked language modeling. It demonstrates superior performance on tasks like: GLUE, SQuAD, and RACE. Under equivalent experimental settings, XLNet consistently beats BERT across a range of 20 NLP tasks, achieving notably superior results. These tasks encompass text classification, question answering, natural language inference, sentiment analysis, document ranking, etc.

XLNet represents a significant advancement in natural language processing by combining the strengths of autoregressive models with innovative training objectives like permutated language modeling. This allows it to capture bidirectional context more effectively than previous models, leading to superior performance on a variety of NLP tasks. The XLnet model is also chosen as one of the foundation models to detect insider threats in this study.

4.2.3.4 DistilBERT

DistilBERT is a smaller, faster, and lighter version of BERT designed to reduce the size and computational requirements of the original BERT model while maintaining most of its performance capabilities [11]. It was developed by the team at Hugging Face in 2019 and represents a significant step towards making transformer-based models more accessible and practical for a wide range of applications:

1. **Knowledge Distillation:** DistilBERT is trained using a technique called knowledge distillation, where a smaller "student" model learns to mimic the behavior of a larger "teacher" model (in this case, BERT). The student model is trained to reproduce the output logits (predictions before softmax) of the teacher model, as well as to match the internal representations (hidden states) learned by the teacher. The student model is also trained on the original masked language modeling task used in BERT to ensure it learns to predict masked tokens correctly. A triple loss integrating language modeling, distillation, and cosine-distance losses was introduced.

2. **Reduced Size:** DistilBERT has about 40% fewer parameters than BERT. DistilBERT retains the core architecture of BERT. but with fewer layers Specifically, it uses 6 transformer layers instead of 12 in the BERT-base model, making it roughly half the size of the original BERT. Each layer is a transformer encoder block that uses self-attention mechanisms to process the input text

3. **Faster Inference:** Because of its smaller size, DistilBERT runs approximately 60% faster than BERT during inference, making it more suitable for deployment in real-time applications where latency is critical.

4. **Comparable Performance:**o Despite its reduced size, DistilBERT achieves around 97% of BERT's performance on various NLP tasks. This includes tasks like text classification, NER, and question answering, among others.

Like its larger equivalents, DistilBERT can be fine-tuned for a wide array of tasks NLP tasks: Text Classification, Named, NER, Question Answering, Text Summarization, and Text Generation etc. DistilBERT is a highly efficient and practical version of BERT that offers a good balance between model performance and computational efficiency. Its reduced size and faster inference times make it an attractive option for deploying NLP models in production environments, especially where resources are limited or where real-time processing is required like edge device computation or intrusion detection. By

leveraging knowledge distillation, DistilBERT maintains a high level of performance while significantly reducing the resource requirements of the original BERT model. This study also adopts the DistilBERT model as one of the foundation models to detect insider threats.

4.2.3.5 Summary

Table 4.1 compares the size, training time, claimed performance, pre-training data, and methodologies of four models. To sum up, the BERT model is the de facto baseline of Pre-trained LLM in deep learning research. The RoBERTa model is well-known for its prediction performance. The XLNet model's structure is more complex and its permutation-based training can learn long-term dependencies well. The DistilBERT model has a simpler structure and a faster inference. These are the reasons that this study chooses the four models as the foundation models of insider threat detection.

4.2.4 Federated PETuning

This paper denotes a client as k and K clients are assumed in the Federated Learning system. Each client can only access their private dataset D_k:= $\{(x_i, y_i)\}$. (x_i, y_i) is the i-th example where x_i and y_i represent the i-th model input and its related label. n_k is the number of examples in D_k while n is the number of examples of all clients' datasets. The client's model parameters are composed of two components: the fixed pre-trained backbone with parameter W_f and the lightweight trainable component with parameter W_t. Therefore, the classification loss of the prediction on example (x_i, y_i) can be represented as $\mathrm{L}(x_i, y_i; W_t, W_f)$. The cross-entropy loss is adopted in this study to train client models in FL. The average loss value of client k can be computed as the below:

$$L_k(W_t, W_f) = \frac{1}{n_k} \sum_{i=1}^{n_k} L(x_i, y_i; W_t, W_f) \tag{4.2}$$

Table 4.1: The Comparison of Foundation Models of Insider Threat Detection

Model	**BERT**	**RoBERTa**	**XLNet**	**DistilBERT**
Size (MB)	Base:130 Large:340	Base:110 Large:340	Base: about 110 Large: about 340	Base: 66
Training Time	Base:12 days(8X V100) Large:4 days(64 TPU chips) or 1 day (280 X V100)	Large:1 day(1024X V100); 4-5 times more than BERT	Large:2.5 days (512 TPU chips); 5 times more than BERT	Base:3.5 days (8 X V100); 4 times less than BERT
Claimed Performance	Outperforms existing models in Oct 2018	2-20% enhancement over BERT	2-15% enhancement over BERT	3% degeneration from BERT
Training Data	16 GB BERT data(Books Corpus + Wikipedia). 3 Billion words	160 GB(16GB BERT data + 144GB additional)	BaseL 16 GB BERT data Large: 113 GB (16 GB BERT data + 97 GB additional). 33 Billion words	16GB BERT data. 3.3 Billion words
Method	Bidirectional Transformer with MLM and NSP	BERT without NSP	Bidirectional Transformer with Permutation based modeling	BERT Distillation

Hence, the overarching objective of the FL system can be articulated as minimizing the weighted average loss of all participated clients' insider threat detection:

$$\arg\min_{W_t, W_f} L(W_t, W_f) = \sum_{k=1}^{K} \frac{n_k}{n} L_k(W_t, W_f) \tag{4.3}$$

Our research focuses on a classic FL system, which involves a central server tasked with coordinating participating clients for their local training and disseminating shared model parameters. Rather than transmitting all bulky Pre-trained LLM parameters, this paper adopts Parameter Efficient Tuning Methods to interchange only lightweight trainable parameters. Before the training begins, our system initializes the backbone of Pre-trained LLMs with frozen parameter W_f and the Parameter Efficient Tuning module named Delta model with trainable parameters W_t. Subsequently, each participant client's local tuning and then the global aggregation on the central server are executed in succession:

Client Local Tuning: upon receiving the trainable parameters from the central server, the participant clients build a complete client model using both lightweight trainable parameters W_t and local Pre-trained LLM's frozen parameters W_f. Subsequently, these client models are trained on their individual local data D_k. When local training is completed, each client k $\in$ [1, K] sends its modified trainable parameters $W_t^{k,r}$ to the central server for federated accumulation.

Server Global Aggregation: The central server disseminates the trainable parameter W_t to the selected K clients. When clients' local tunings are completed, the central server will receive updated trainable parameters $W_t^{k,r}$ from clients. Subsequently, the server executes federated aggregation of these received clients' updated trainable parameters, updating the W_t^r by incorporating these updates:

$$W_t^r = \sum_{k=1}^{K} \frac{|D_k|}{\sum_{k=1}^{K} |D_k|} W_t^{k,r} \tag{4.4}$$

Where $|D_k|$ is the size of the client k's local dataset and r is the global epoch number. After that, the server returns the updated trainable parameters W_t^r to the clients. Apart

from FedAvg, FedITD is also open to other fusion algorithms, such as Adaptive Federated Averaging [72], Shuffle Iterative Average [73], etc., against data inversion attack or data poisoning.

The training process outlined above iterates until a specific criterion is satisfied, such as meeting the convergence condition or reaching the maximum number of global communication rounds R. Homomorphic encryption is applied in data exchange to avoid any sensitive or privacy information leakage. This entire process is outlined in Figure 4.3 Algorithm 1 by using pseudo code.

4.2.5 PETuning Methods of PLMs

1. **Addition-based approaches** introduces additional trainable neural modules or parameters that are not present in the original model or process into the Pre-trained LLMs. This study chooses the adapter-based tuning method as the representative of addition-based approaches in the experiment. Houlsby et al. [13] proposed inserting small modules, known as adapters, between transformer layers in 2019. Typically, the adapter layer employs a down-projection with $W_d \in R^{d \times m}$ to map the input $h \in R^d$ into a smaller-dimensional space identified by a bottleneck dimension m. Following this, a nonlinear activation function $f(\cdot)$, and an up-projection with $W_u \in R^{m \times d}$ map it back to the input size d. As Figure 4.4 illustrates, these adapters are enveloped by a residual connection, resulting in the final formula:

$$h = f(hW_d)W_u + h \tag{4.5}$$

Typically, two adapters are inserted in sequence: one behind the multi-head attention while the other one after the Feed Forward Network (FFN) sub-layer. During fine-tuning, only adapter layers $W_t = \{W_u, W_d\}$, the normalization layer's parameters, and the final classification layer's parameters are trainable, constituting around 0.5% to 8% of the parameters of the complete model in the tuning process, meanwhile retaining the parameters of the Pre-trained LLM frozen. The total

Input: Client set C; Global epoch number R; Local epoch number E; Pre-trained Language models' original parameters W_f; Local trainable and efficient parameters W_t ;The K clients are indexed by k and the local dataset D_k belongs to the k-th client; Local PETuning Method T.
Output: a fine-tuned global model W_{R+1}
Before Training: Initialize W_t^0 and W_f on the central server and every client in C.
Server Global Aggregation:
For each global round $r = 1$ to R do
  The global model W_t^{r-1} is shared to each client.
   For each client $k \in C^t$ in parallel do
     $W_t^{k,r} \leftarrow$ Client Local Tuning (k, W_t^{r-1})
   End
Get local updated parameters $W_t^{k,r}$
Conduct global aggregation $W_t^r = \sum_{k=1}^{K} \frac{|D_k|}{\sum_{k=1}^{K} |D_k|} W_t^{k,r}$
End
Return W^R

Client Local Tuning (k, W_t^r):
$W^r \leftarrow$ Load and assemble W_t^r and W_f
For each epoch $e = 1$ to E do
$W_t^{k,r+1} \leftarrow T(D_k, W_t^r)$
End
Transmit $W_t^{k,r+1}$ to the server

Figure 4.3: Algorithm 1: Training Procedure of Federated Parameter Efficient Tuning

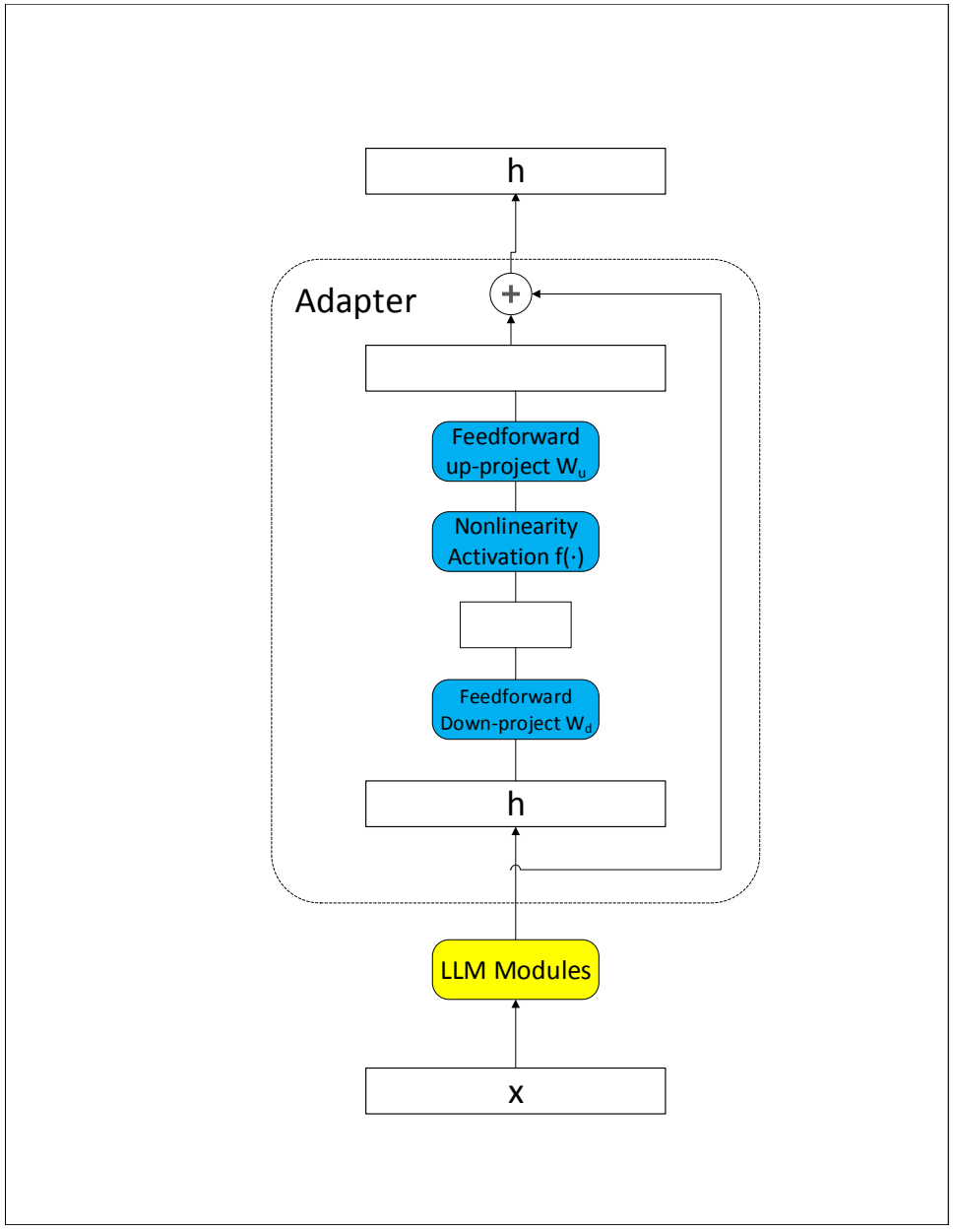

Figure 4.4: The Architecture of the Adapter Module [13]

number of parameters added per layer is 2md +d +m.

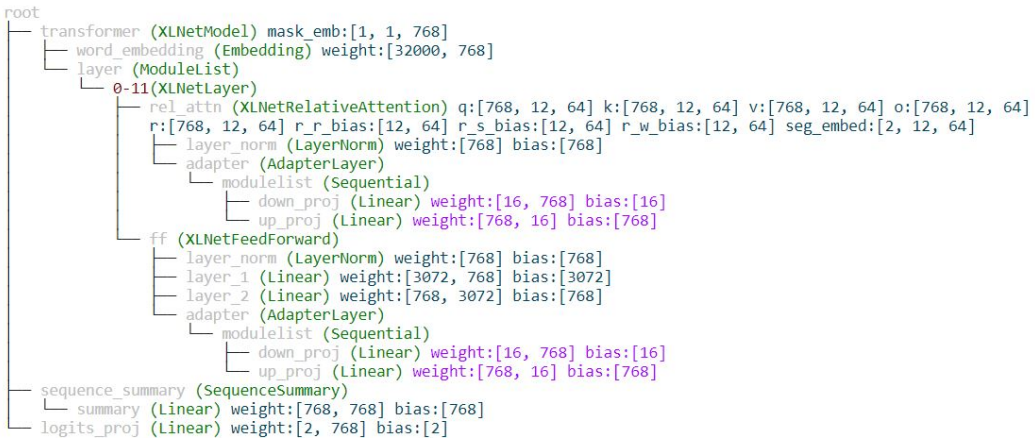

Figure 4.5: The Original Architecture of XLNet with Adapter Tuning

Note:the purple part is the delta parameters inserted into the backbone model; the blue part is the tunable parameters of the backbone model.

As per the original paper [13] (Figure 4.5), we inserted two adapters sequentially into the XLNet model: one following the multi-head attention while the other after the FFN sub-layer. The experiment results show that Macro Average F1 is 0.9286 and accuracy is 99.17%. We also explored another alternative method (Figure 4.6): only one adapter is added after the FFN "add & layer norm" sub-layer. However, Macro Average F1 declines to 0.9027 and accuracy decreases to 98.80%. Therefore, we adopt the former.

2. **Specification-based methods** entail designating a small part of parameters within the original model or process as trainable, while the remainder is set to remain frozen. This thesis selects BitFit as the representative of the specification-based method in our experiment. Zaken et al. [15] introduced BitFit and made the experiment that by freezing all other parameters and solely tuning the bias terms of the pre-trained LLM, i.e., $W_t = \{b_{(\cdot)}^{l,(\cdot)}\}$, which constitute only 0.08% and 0.09% of the total number of parameters in BERT base and BERT large models respectively, while competitive performance can still be achieved compared to full-tuning

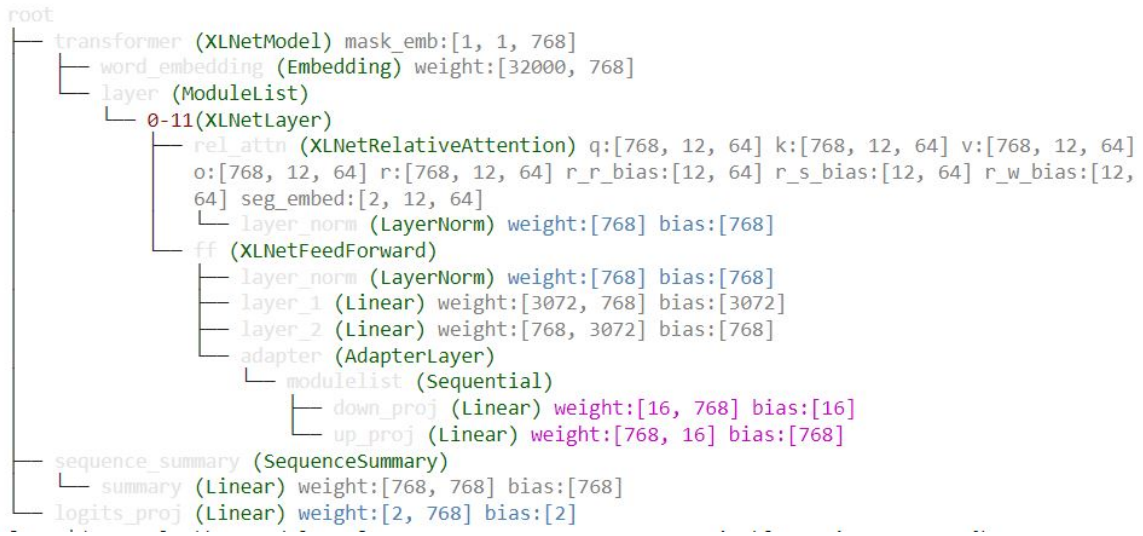

Figure 4.6: The Alternative Architecture of XLNet with Adapter Tuning

Note:the purple part is the delta parameters inserted into the backbone model; the blue part is the tunable parameters of the backbone model.

the entire model.

$$f(X) = f(X) + B \tag{4.6}$$

where f is all functions and B is the bit term. The number of parameters is L×$(7\times d_h + d_m)$. d_h denotes the hidden dimension of the transformer model while d_m stands for the intermediate dimension between up mapping and down mapping. As per the original paper [15], the bias term is either added or unfreezed in the following modules: Attention layer encompassing all positions q, k, and v, along with the output linear layer, Feed Forward network, normalization layer, and Classifier. Figure 4.7 illustrates how to implement the Bitfit tuning method on the RoBERTa model:

3. **Reparameterization-based methods** convert current parameters into a more parameter-efficient form through reparameterization. This study chooses LoRA as the representative of the reparameterization-based method in our experiment. Hu et al. [14] introduced trainable low-rank matrices (LORA) alongside multi-head attention in 2021, aiming to convert original parameters into a more parameter-

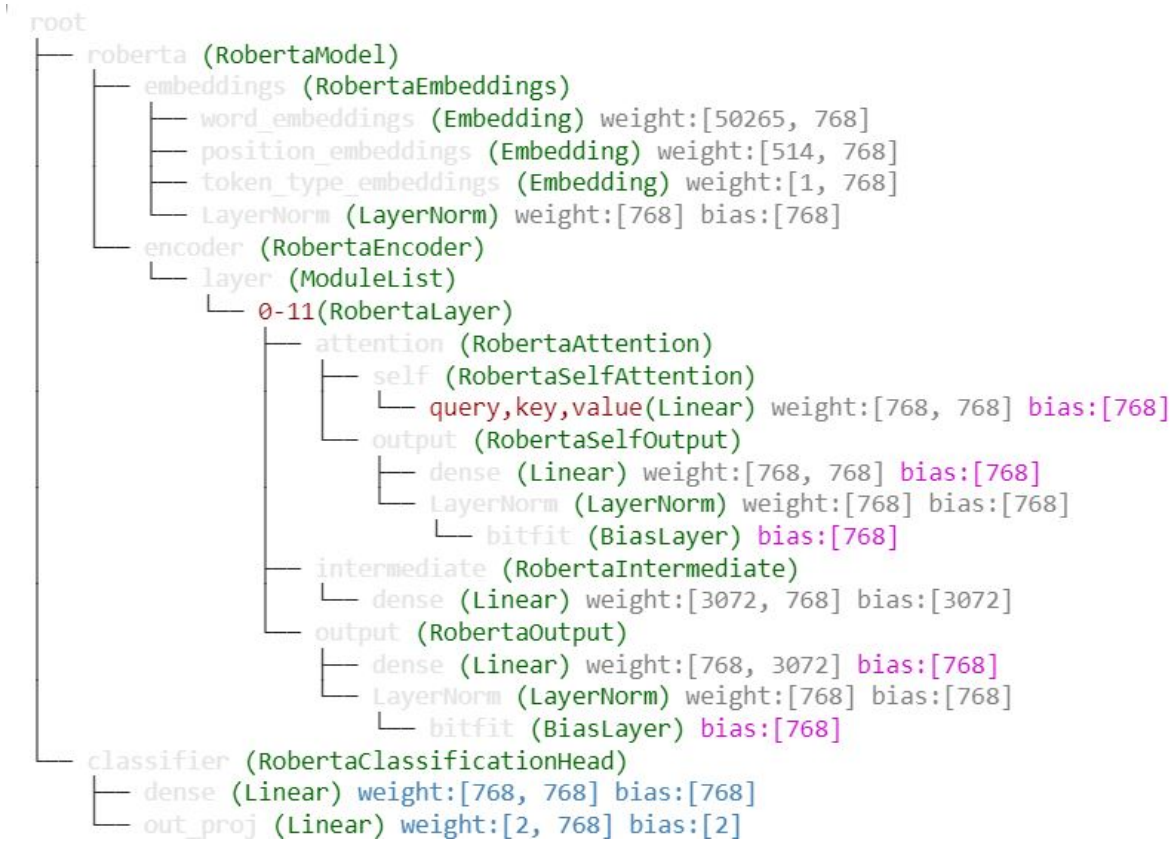

Figure 4.7: The Architecture of RoBERTa with BitFit Tuning

Note: the purple part is the delta parameters inserted into the backbone model; the blue part is the tunable parameters of the backbone model.

efficient style. For a pre-trained weight matrix $W_0 \in R^{d\times k}$, LoRA express its update using a low-rank decomposition

$$W_0 + \Delta W = W_0 + BA \tag{4.7}$$

Where B $\in R^{d\times r}$ and A $\in R^{r\times k}$, with the rank r $\ll$ min(d,k). For a particular input x to the linear projection in multi-head attention, LoRA adjusts the projection output h as follows:

$$h = W_0 x + \Delta W x = W_0 x + sBAx \tag{4.8}$$

where s is a tunable scalar hyperparameter that is larger or equal to 1. As Figure 4.8 shows, LoRA employs this modification to the query and value projection matrices $(W_q; W_v)$ within the multi-head attention sub-layer. During training, W_0 remains fixed and doesn't get gradient updates, whereas A and B encompass trainable parameters. This approach enables LoRA to enhance training efficiency with less than 1% of trainable parameters $W_t = \{B, A\}$, while still achieving performance comparable to fine-tuning on the GLUE benchmark. They testify to the efficacy of their approach across pre-trained language models of different scales and architectures.

Initially, this study adopted the original paper's method [14]: For example, when the foundation model is DistilBERT, as Figure 4.9 shows, q and v matrixes in the attention layer are changed with LoRA matrixes. However, we found that revising other linear layers can result in better performance. As Figure 4.10 illustrates, FFN's linear layers, preclassifier layer, and classifier layer are modified with LoRA matrixes. In our implementation, we opted not to substitute the linear layer of the backbone model with the linear layer from loralib. Instead, we introduced a parallel module into the backbone architecture. In essence, we regard (W_0 + sBA)x as W_0 x + sBAx, incorporating sBAx as a parallel insertion module. LoRA modules are then added after every linear layer. Using the original method, the Macro Average

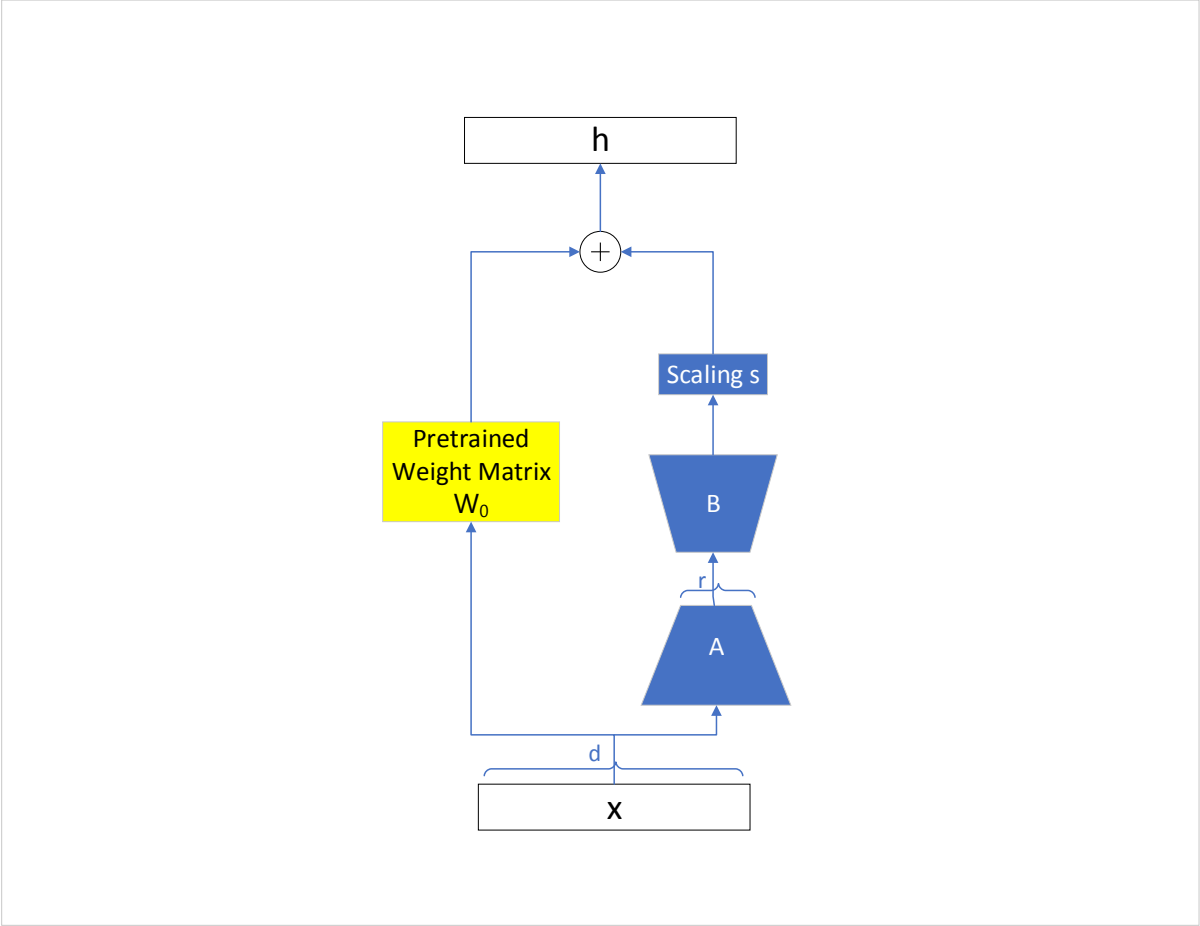

Figure 4.8: The Architecture of LoRA Module

```
root
├── distilbert (DistilBertModel)
│   ├── embeddings (Embeddings)
│   │   ├── word_embeddings (Embedding) weight:[28996, 768]
│   │   ├── position_embeddings (Embedding) weight:[512, 768]
│   │   └── LayerNorm (LayerNorm) weight:[768] bias:[768]
│   └── transformer (Transformer)
│       └── layer (ModuleList)
│           └── 0-5(TransformerBlock)
│               ├── attention (MultiHeadSelfAttention)
│               │   ├── q_lin,v_lin(Linear) weight:[768, 768] bias:[768]
│               │   │   └── lora (LowRankLinear) lora_A:[8, 768] lora_B:[768, 8]
│               │   └── k_lin,out_lin(Linear) weight:[768, 768] bias:[768]
│               ├── sa_layer_norm,output_layer_norm(LayerNorm) weight:[768] bias:[768]
│               └── ffn (FFN)
│                   ├── lin1 (Linear) weight:[3072, 768] bias:[3072]
│                   └── lin2 (Linear) weight:[768, 3072] bias:[768]
├── pre_classifier (Linear) weight:[768, 768] bias:[768]
└── classifier (Linear) weight:[2, 768] bias:[2]
```

Figure 4.9: The Original Architecture of DistilBERT model with LoRA Tuning

F1 is 0.9390. In comparison, the new method can improve Macro Average F1 to 0.941 while accuracy is the same. Therefore this study adopts the new method.

Figure 4.10: The Alternative Architecture of DistilBERT model with LoRA Tuning

4.2.6 Transfer Learning

When the federated learning is completed, we can get the global model. However, if we utilize the global model directly on the client's local data, it may still exhibit poor performance for the specific client. The federated learning system operates well under the assumption that the training and test data are independent and identically distributed (i.i.d.). However, this assumption rarely holds in practice due to variations in data among different clients, over time, and across different locations. Consequently, the global model trained by the federated learning system often struggles to effectively handle the distribution disparity between the training data and the client's local data, a phenomenon well-known as **domain shift**. This discrepancy happens from the distribution disparity between the client's data and training data. The global model hosted on the central server typically captures only coarse features from all clients, thus struggling to grasp

the nuanced, fine-grained information specific to individual clients. On the other hand, the local data from individual clients typically lacks anomaly samples and thus it is difficult to train a robust model. Therefore, to achieve excellent customized client models, it is necessary to apply transfer learning to fill this gap rather than collecting labeled local data and training a new classifier for every client. Moreover, the target domain data is usually unlabeled, necessitating unsupervised adaptation techniques rather than supervised adaption techniques to transfer the learning we have already got.

Deep CORAL [56] is an excellent unsupervised adaption technique to incorporate learning a nonlinear transformation aimed at minimizing the difference in feature covariances between source and target domains. We have source-domain training examples $D_S = \{x_i\}, x_i \in R^d$ with labels $L_S = \{y_i\}$, $i \in \{1, \cdots, L\}$, and unlabeled target data $D_T = \{u_i\}, u_i \in R^d$. Let n_S and n_T stand for the number of source and target data, correspondingly. C_S indicates the feature covariance matrix of the source data while C_T represents the feature covariance matrix of the target data.

The differential CORAL loss is designated as the discrepancy between the covariance of the source and target features.

$$L_{CORAL} = \frac{1}{4d^2}||C_S - C_T||_F^2 \tag{4.9}$$

where $||\cdot||_F^2$ represents the squared matrix Frobenius norm. The covariance matrices for the source and target data are represented respectively as follows:

$$C_S = \frac{1}{n_S - 1}(D_S^T D_S - \frac{1}{n_S}(\mathbf{1}^T D_S)^T(\mathbf{1}^T D_S)) \tag{4.10}$$

$$C_T = \frac{1}{n_T - 1}(D_T^T D_T - \frac{1}{n_T}(\mathbf{1}^T D_T)^T(\mathbf{1}^T D_T)) \tag{4.11}$$

Our goal is that the ultimate deep features must possess both discriminative powers to effectively train a robust classifier and constant to differences between the source and target domains. Solely minimizing the classification loss may result in overfitting to the source domain, thereby diminishing performance on the target domain. Conversely,

only minimizing the CORAL loss alone could produce deteriorated features and poor detection performance. Training jointly with minimizing both the classification loss and CORAL loss can result in learning features that are effective on the target domain.

$$L = L_{CLASS} + \sum_{i=1}^{t} \lambda_i L_{CORAL} \tag{4.12}$$

Here, t represents the number of CORAL loss layers in a deep network, and λ is a weight parameter that balances the adaptation with classification accuracy on the source domain. In this study, the covariance matrices for the source and target features are calculated on the temporal sequence feature. CORAL Loss is then seamlessly integrated into the total loss of the pre-trained LLM instead of adding an alignment layer. The client model is re-trained with the target of minimizing the total loss L including both the classification loss and CORAL loss on the client's local data. By tuning λ, these two losses compete with each other and finally converge to an optimal equilibrium during training, resulting in final features that are anticipated to perform effectively on both the target domain and source domain. The detailed procedure is described in 4.11 Algorithm 2. Besides Deep CORAL, FedITD offers extensibility by accommodating other transfer learning algorithms.

4.3 Experiment

Two kinds of experiments are carried out to assess the effectiveness of the framework FedITD. The first kind of experiment aims to assess federated PETuning methods including their fundamental capability of insider threat detection, resource costs, and privacy preserving. The other one is to evaluate the effectiveness of transfer learning consisting of unsupervised domain adaptation's performance evaluation, comprehensive ablation analysis of FL and TF's contribution, performance evaluation on the highly heterogeneous and slightly heterogeneous target domain, generalization ability evaluation, and how to attain the optimal equilibrium that performs well across source domain, target domain (unlabeled local data), and global test.

Input: Client set C; Local epoch number E; a fine-tuned global model f_S; The K clients are indexed by k and the local dataset D_k belongs to the k-th client; the global test data set are original data including all anomaly scenarios; the source and target validation dataset are the augmented data that do not join model training in the source and target domain. λ is a weight parameter to tradeoff between classification performance and domain adaption.
Output: a well-customized client model f_k for client k
Download the well-tuned global model f_S from the central server
Repeat
Tune the value of λ
For each epoch $e = 1$ to E **do**
Calculate classification loss by using cross-entropy
Calculate CORAL loss by using (4.9)
Train client k's model f_k on the private data set D_k using (4.12)
Evaluate the client model f_k's performance on the source validation dataset
Evaluate the client model f_k's performance on target validation dataset
End
Evaluate the client model f_k's performance on the global test dataset
Until Find an optimal λ to achieve optimal equilibrium that the model performs the best across source validation dataset, target validation dataset, and global test dataset

Figure 4.11: Algorithm 2: Training Procedure of Federated Transfer Learning

4.3.1 Implementation Details

4.3.1.1 Experiment Setting

This study conducts our experiments utilizing Google Colab, a virtual environment equipped with an NVIDIA A100-SXM4-40GB GPU, 83.48GB of RAM, and a disk capacity of 166.77 GB. PyTorch version is 2.0.0 and Python version is 3.10.11. Key software packages utilized for model development include PyTorch, Pandas, Matplotlib, Transformers, and Scikit-learn. This study adopts RoBERTa-Base, BERT-base-cased, XLNet-base-cased, and distilbert-base-cased-distilled-squad as the foundation models using the AutoModelForSequenceClassification package released by Huggingface V4.39.3. All parameter-efficient tuning approaches are implemented based on OpenDelta [74], which is an Open-Source toolkit for parameter-efficient tuning. The infrastructure of the Federated Learning system is developed based on FedLab [75], which is a flexible federated learning framework. Throughout the study, our model was trained with batch sizes of 32. In federated learning, the model is tuned for 1 local epoch in each client's site, while the global communication epoch is 10 rounds. Cross-entropy loss and the Adam optimizer are employed for local training. For each model and tuning approach, we conduct a hyperparameter sweep. The best model hyperparameters are chosen based on the metric performance on the test data set. Specifically, we explore learning rates from 1e-6, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3, 1e-2. For the Adapter, we search the bottleneck dimensions from 16, 64, and the ranks from 4, 8, 16 are examined for LoRA.

4.3.1.2 Dataset

This study chooses the CERT r6.2 dataset as the experimental data source because it is the most prominent open dataset for insider threat detection. Compared with CERT r4.2, CERT r6.2 is a sporadic dataset containing much fewer malicious users and their behavior data but much more users' normal behavior data. Thus Detecting insider threats within such a highly imbalanced dataset is considerably challenging. This dataset is introduced

in detail in section 3.3.1. The details of data processing are described in Section 3.3.2 and 3.3.3. This study addresses highly imbalanced data issues through language data augmentation techniques to upsample minorities, including contextual embedding word insert and substitution predicted by associate LLMs, and contextual sentence embedding predicted by the pre-trained transformer model the GPT-2 model.

4.3.1.3 Evaluation Metrics

To assess the effectiveness of our model, we employed the following performance metrics as evaluation criteria: accuracy, precision, recall, F1 score, and AUC. They are introduced in detail in section 3.5.1.3. Because the global test dataset contains only a few anomaly instances and a huge number of normal instances. If the model predicts all instances to be normal, it can still achieve high accuracy. Therefore, accuracy is not a reliable measure metric in this experiment. The F1 score provides a comprehensive measure of both precision and recall. However, if the F1 score of one class is increased, there may be a decrease in the F1 score of the other class. Macro Average F1-score takes the F1 scores of each class and averages them, treating all classes equally. Therefore, this study chooses the Macro Average F1-score as the first measure metric and keeps accuracy as the second measure metric.

Additionally, we also used the following performance metrics as evaluation criteria:

- **Communication Cost**

  The communication cost 'c' can be computed using the formula:

  $$c = r \times n \times s \times 2 \tag{4.13}$$

  where r is the global communication round, n is the count of the clients, and s is the size of the trainable parameters. The trainable parameters include tunable parameters in both the tunable modules and the backbone model.

- **Storage Overhead**

  Storage overhead is measured by the model size. The size of the model includes

the size of the backbone models and the size of the delta model. The latter is the size of a tunable module, e.g., Adapter, or RoLA.

- **Memory Usage**

  Memory usage is the amount of RAM required to run the model during inference. It is a very important performance metric and is especially useful for deploying models on devices with limited memory.

4.3.2 Evaluation and Analysis

4.3.2.1 Evaluation Result

1. **Federated Tuning Performance**

   The experiments are performed in the Cross-silo Setting. **Cross-silo Setting** caters to multiple users, typically no more than 100 clients. Almost all experiments except for evaluating the impact of a large number of clients are performed in this setting. The total number of clients is set to 10. The central server chooses all clients for training in every communication round. In the experiment of federated learning, this study adopts contextual word embedding using the associate LLMs to upsample the anomaly behavior data because this method is proven to make the model perform better [12]. In the augmented dataset, the ratio of original normal user behavior data to augmented anomaly user behavior data is 55%: 45%. 80% of the whole data is split as the training dataset, 10% of the whole data is split as the validation dataset, and the remaining 10% is kept as the test dataset. The ratio of original normal user behavior data to augmented anomaly user behavior data is 1:1 in the both training dataset and the validation dataset. In the test dataset, both normal and anomaly user behavior data are all original data. Then the training dataset and validation dataset are divided among clients as their local data.

   As depicted in Table 4.2, in terms of federated learning with PETuning methods, the RoBERTa model with LoRA tuning demonstrates superior performance com-

Table 4.2: The Performance of Federated and Central Tuning of Pre-trained LLMs

Macro AVG F1 (Accuracy)	RobertTa	BERT	XLNet	DistilBert	AVG
FedFT	0.9216(0.9917)	0.941(0.9935)	0.941(0.9935)	**0.9534(0.9954)**	0.9393(0.9935)
FedAP	**0.9369(0.9935)**	0.9101(0.9898)	0.9286(0.9917)	0.917(0.9908)	0.9232(0.9915)
FedBF	**0.9216(0.9917)**	**0.8906(0.9871)**	**0.7175(0.9445)**	0.9034(0.9889)	0.8583(0.9781)
FedLR	**0.945(0.9945)**	0.9241(0.9917)	0.9286(0.9917)	0.941(0.9935)	**0.9347(0.9929)**
CenFT	**1(1)**	0.9633(0.9963)	0.9739(0.9972)	0.9534(0.9954)	0.9727(0.9972)
CenAP	**0.9739(0.9972)**	0.9633(0.9963)	0.9502(0.9945)	0.9101(0.9898)	0.9494(0.9945)
CenBF	**0.9739(0.9972)**	0.945(0.9945)	0.8451(0.9787)	0.9314(0.9926)	0.9239(0.9908)
CenLR	0.945(0.9945)	0.955(0.9954)	**0.9739(0.9972)**	0.9534(0.9954)	**0.9568(0.9956)**

Note: Accuracy is in the bracket while macro average f1 is outside of the bracket. The bold font is the best performer of the same model using the same tuning method. The blue font is underperformers. FedFT: Federated Full Tuning; FedAP: Federated PETuning using Adapter; FedBF: Federated PETuning using BitFit; FedLR: Federated PETuning using LoRA; CenFT: Central Full Tuning; CenAP: Central PETuning using Adpater; CenBF: Central PETuning using BitFit;CenLR: Central PETuning using LoRA

pared to other methods, while the XLNet model with BitFit tuning performs the poorest. Across various base models, RoBERTa consistently exhibits the highest performance, outperforming DistilBert, BERT, and XLNet with all PETuning methods. The performance ranking is as follows: RoBERTa > DistilBert > BERT > XLNet. When comparing different PETuning methods, the performance rankings are as follows: LoRA > Adapter > BitFit. In terms of federated learning with full- tuning methods, DistilBERT performs best with Macro Average F1 0.9534 and an accuracy of 99.54%. This is consistent with our past conclusion that the simplified encoder architecture of the transformer model performs the best in insider threat detection [12]. Only the Macro Average F1 score of XLNet with BitFit tuning and BERT with BitFit tuning are lower than 0.90. It shows that although the BitFit module is added or unfreeze bit term in many layers of the foundation model, it carries much less weight in classifying insider behavior sequences.

In terms of central learning with PETuning methods, the RoBERTa model with adapter tuning and BitFit tuning and the XLNet model with LoRA tuning demonstrates superior performance compared to other methods, while the XLNet model with BitFit tuning performs the poorest. Across various base models, RoBERTa consistently exhibits the highest performance, outperforming DistilBert, BERT, and XLNet in all PETuning methods. The performance ranking is as follows: RoBERTa > BERT > DistilBert > XLNet. When comparing different fine-tuning methods, the performance rankings are the same as federated learning: LoRA > Adapter > BitFit.

Relative Percentage of Macro Average F1 between federated learning and central tuning is the macro average F1 of the federated tuning minus the macro average F1 of central tuning with the same tuning method and then divided by the macro average F1 of the central tuning with the same tuning method. As Table 4.3 shows, XLNet with BitFit tuning performs the worst as the reduction rate attains 15.10%. Additionally, the relative percentage of macro average F1 of RoBERTa with full-

Table 4.3: The Relative Percentage of Macro Avg F1 between Federated and Central Tuning of Pre-trained LLMs

Gap	RobertTa	BERT	XLNet	DistilBert
FT	**-7.84%**	-3.38%	-3.38%	0.00%
Adapter	-5.70%	-5.52%	-4.65%	0.76%
BitFit	-5.37%	-5.76%	**-15.10%**	-3.01%
Lora	0.00%	-4.65%	-4.65%	-1.51%

tuning is -7.84%. However, all other models with other tuning methods perform well as their relative macro average F1 percentage reductions are all less than 6%. Especially DistilBERT with full-tuning and RoBERTa with LoRA tuning's macro average F1 have no changes between federal tuning and central tuning.

Table 4.4: The Relative Percentage of Macro Avg F1 between Federated PETuning and Federated Full-Tuning of Pretrained LLMs

Gap	RobertTa	BERT	XLNet	DistilBert
FedFT	-3.34%	-1.30%	-1.30%	0.00%
FedAdapter	-1.73%	-4.54%	-2.60%	-3.82%
FedBitFit	-3.34%	**-6.59%**	**-24.74%**	-5.24%
FedLora	-0.88%	-3.07%	-2.60%	-1.30%

The relative percentage of Macro Average F1 between federated PETuning and federated full-tuning is the percentage of performance gap achieved by federated PETuning relative to the best federated full-tuning performer DistilBERT in terms of Macro average F1. As Table 4.4 shows, XLNet with BitFit tuning performs the worst as the gap is -24.74%. Additionally, the relative macro average F1 percentage

of BERT with BitFit tuning is 6.59%. However, all other models with other tuning methods perform well as their relative percentages of macro average F1 are all less than 6%.

From the above results, we can see that the performance of most PETuning methods downgrades by less than 6% in federated learning environments compared to corresponding PETuning methods in the central learning environments. Additionally, the performance of most PETuning methods downgrades less than 6% in federated learning environments compared to the best full-tuning methods in the same environment. However, FL using Pre-trained LLMs with most PETuning methods can achieve high Performance in terms of Macro Average F1 score (>0.90) and accuracy. RoBERTa performs better than other models and exhibits greater robustness in a federated learning environment. LoRA is the most effective PETuning method when leveraging a strong Pre-trained LLM like RoBERTa or DistilBert. Nevertheless, some combinations of Pretrained LLMs and PETuning methods especially XLNet with BitFit tuning method have a considerable degeneration compared with the best federated learning full tuning method (-24.74%) or corresponding PETuning method in the central learning environment (-15.10%).

Finally, we evaluated the impact of many clients in the Large-scale Cross-device Setting. **Large-scale cross-device setting** represents the other federated setting intended for deployment across a large number of clients. In this scenario, data held by local clients are more sporadic than in the Cross-silo Setting. In the experiment, the total number of clients in the large-scale settings is 100, and the sampling rate is set as 10%. In each communication round, 10 clients are randomly sampled from all available clients to participate in both local training and federated aggregation processes in the federated learning process. Employing RoBERTa as the base model and LoRA as the tuning approach, we replicate the aforementioned training and testing procedures. The findings indicate that the algorithms' performance remains robust. The Macro Average F1 experiences only a minor decrease of 3.34%, while

the accuracy decreases by a mere 1.15%. This suggests that the number of clients has minimal influence on performance.

2. **Resource Cost**

   The resources associated with various PETuning methods encompass the storage overhead, the communication cost, and the memory usage.

Table 4.5: The Size of Trainable Parameters of Fine Tuning Methods

The Size of Trainable Parameters (MB)	RobertTa	BERT	XLNet	DistilBert
FedFT	124.64(100%)	108.31(100%)	117.31(100%)	65.78(100%)
FedAP	1.2(0.96%)	**1.74(1.64%)**	0.62(0.55%)	0.29(0.46%)
FedBF	0.66(0.53%)	0.08(0.08%)	0.06(0.06%)	**0.04(0.07%)**
FedLoRA	0.89(0.71%)	0.29(0.27%)	0.70(0.62%)	0.37(0.58%)

Note: The size of the trainable parameter is outside of the bracket while the trainable ratio is in the bracket

Table 4.5 shows the size of trainable parameters and the trainable ratio which is the ratio of the size of trainable parameters to the size of the corresponding models' parameters. Observations reveal that the size of DistilBERT with the BitFit tuning method is remarkably compact, occupying only 0.04MB, which represents a mere 0.07% of the backbone model's size. In contrast, the BERT model with Adapter tuning exhibits the largest size at 1.74MB, attributed to the insertion of learnable modules, termed Adapters, into certain transformer layers. According to Table 4.5, in terms of tuning methods, the sizes of their trainable parameters are ordered as follows: FedFT $\gg$ FedAP $>$ FedLR $>$ FedBF. Regarding foundation models, the sizes of their trainable parameters are ordered as RoBERTa $>$ BERT $>$ XLNet $>$ DistilBERT. The experimental findings reveal that the size of trainable parameters

only accounts for 0.04% to 1.64% of the size of all parameters of the models. That means that PETuning methods can use 62 times to 1955 times fewer trainable parameters than a full-tuning method to attain a performance close to the full-tuning method in FL settings. In addition, by deploying multiple tasks on a local client, PETuning methods can facilitate different tasks to share one Pre-trained LLM. The client is allowed to retrain only a small number of trainable parameters for each task, thereby reducing storage requirements. Moreover, this reduction in the size of trainable parameters results in a significant decrease in communication cost because the amount of data in the communication significantly reduces. Furthermore, it is very helpful to preserve client privacy since the client data exposed to the network is much decreased.

Table 4.6: The Communication Cost of Fine Tuning Methods

The total communication cost in the traning(MB)	RobertTa	BERT	XLNet	DistilBert
FedFT	24928	21662	23462	13156
FedAP	240	348	124	58
FedBF	132	16	12	8
FedLoRA	178	58	140	74

Table 4.6 illustrates the communication costs for all tuning methods under FL settings. The communication cost of the DistilBERT model with the BitFit tuning method is notably small, occupying just 8MB. Conversely, the communication cost of the BERT model with Adapter tuning is the largest, measuring 348MB. As shown in Table 4.6, in terms of the PETuning method, the communication costs are ranked as follows: FedFT $\gg$ FedAP $>$ FedLR $>$ FedBF. Regarding foundation models, the communication costs are ranked as RoBERTa $>$ BERT $>$ XLNet $>$

DistilBert. The experimental findings indicate that the Federal PETuning methods greatly enhance training efficiency by substantially decreasing communication costs from 98.39% to 99.94% compared to their FedFT methods on their corresponding models. This reduction in communication costs also leads to a significant decrease in training time during both uploading and downloading parameters. Consequently, Federal PETuning proves to be a more practical choice for real-world applications, especially in networks with communication constraints.

Table 4.7: The Memory of The Delta Model and The Total Model

The Memory of the delta model (the backbone model) (MB)	RobertTa	BERT	XLNet	DistilBert
FedFT	475.50(475.50)	413.18(413.18)	450.31(450.31)	250.95(250.95)
FedAP	2.32(480.14)	6.96(431.58)	2.32(452.15)	1.16(253.27)
FedBF	0.25(475.82)	0.39(413.64)	0.25(447.82)	0.16(251.10)
FedLoRA	1.12(477.75)	1.12(415.43)	2.81(453.13)	1.48(253.90)

Note: The memory of delta model is outside of the bracket while the memory of the backbone model is in the bracket

Memory usage is also an important aspect of the performance. However, most researchers ignored memory usage. Table 4.7 illustrates the memory efficiencies of PETuning methods and the Full-tune method across various models in a Federated Setting. In terms of the PETuning method, the memory usages are ordered as follows: FedFT $\gg$ FedAP $>$ FedLoRA $\gg$ FedBF. Regarding the foundation models, the memory usages are ranked as RoBERTa $>$ XLNet $>$ BERT $\gg$ DistlBERT. The delta model's memories of RoBERTa model with BitFit tuning and XLNet with BitFit tuning are the smallest at 0.25 MB while the delta model's memory of the BERT model with Adapter tuning has the largest memory usage which is 6.96 MB.

PETuning methods' delta model only accounts for 0.05% to 1.61% memory usage compared to the corresponding original foundation models. This advantageous feature is particularly beneficial for local clients in real-world FL systems, especially edge devices.

3. **Privacy Preserving**

   The experiment of an embedding inversion attack is performed to evaluate the privacy protection capabilities of FedITD. The embedding inversion attack is to recover original input from clients' model weight updates in an FL environment. After removing encryption and federated dropout, this study initializes a tensor of random embeddings and uses a backward optimization process to minimize the Mean Square Error (SME) loss between recovered embeddings and actual embeddings [76]. After optimization, the closest token indices are identified by the Euclidean distance and then decoded back to the recovered text. The F1 score is used to evaluate the performance of the inversion attack. We randomly selected 120 daily user behavior samples from the CERT dataset r6.2 after data processing as an attack dataset and evaluated the performance of the attack using the RoBERTa model with full tuning and LoRA tuning methods. The result shows that the F1 score of the former is 0.1803 while the latter is reduced to 0.0602. It demonstrated that FedITD can more effectively protect clients against data reconstruction attacks even if encryption and federated dropout are removed.

4. **Transfer Learning**

   We performed three experiments to assess the efficacy of transfer learning in FedITD. The initial experiment involves assessing the fundamental capability of insider threat detection through TL, the performance comparison of various tuning methods, and an ablation study. The second experiment aims to verify its capacity for generalization, while the third one seeks to validate unsupervised domain adaption's effectiveness and identify its optimal configurations.

1) **Detection Performance Evaluation**

Because XLNet net with BitFit tuning has the worst performance after FL. This study chooses the XLNet model tuned using the BitFit method in FL as the starting point. This study compares FedITD's performance with that of other tuning methods: the **Full-tuning** method entails adjusting all parameters of the downloaded global model from FL using the client's local data without explicitly employing transfer learning techniques, such as minimizing distribution divergence between domains. The **PETuning** method involves continuing parameter-efficient tuning on the downloaded global model obtained from FL using the client's local data, without explicitly employing transfer learning techniques as well. Furthermore, we perform an ablation study to assess the contributions of the two primary elements: federated learning and transfer learning. The term **"Only Fed"** refers to using the global model trained in FL to apply to client local data without a personalized transfer learning process. In addition, **"Only TL"** means that the global model XLNet undergoes retraining solely using client local data with transfer learning, without involvement of federated learning anymore.

To perform the test of transfer learning, we randomly chose 70% of the training data that participated in FL training from each client as the source domain's training dataset. Select 70% of data from each client's validation dataset that did not participate in FL training as the source domain's validation dataset. The ratio of original normal user behavior data to augmented anomaly user behavior data is 1:1 in the source domain's data. Test datasets in the federated tuning experiments are used for global test. One client (Client 1) is then chosen from ten clients and uses its training dataset as the target domain's training dataset while its validation dataset as the target domain's validation dataset, i.e., the local test dataset. Because the target domain's dataset uses the same data augmentation method as the source domain's dataset (inserting or replacing context embedding words predicted by associate LLMs), data heterogeneity is light. To increase data heterogeneity, we also

use the dataset augmented by another method (the context embedding sentences predicted by the GPT-2 model) as the target domain's training and local test dataset. The former is called the Lightly Heterogeneous (LH) dataset while the latter is called the Highly Heterogeneous (HH) dataset.

Table 4.8: The Performance of transfer Learning on LH and HH datasets

	Test	Full-tuning	PETuning	Only Fed	Only TF	FedITD
LH Dataset	Local Test	0.9737(0.9737)	0.9163(0.9165)	0.3391(0.5131)	0.9938(0.9938)	**0.9954(0.9954)**
	Global Test	0.7466(0.9519)	0.5743(0.8327)	0.7175(0.9445)	0.9062(0.9890)	**0.9125(0.9881)**
HH Dataset	Local Test	0.9378(0.9379)	0.8103(0.8105)	0.3571(0.5556)	0.9479(0.9481)	**0.9640(0.9643)**
	Global Test	0.6809(0.9233)	0.5234(0.7717)	0.4930(0.9723)	0.9423(0.9917)	**0.9517(0.9954)**

Note: Accuracy is in the bracket while macro avg f1 is outside of the bracket. The bold font is the best performer.

From Table 4.8, we can see that utilizing the global mode trained in FL directly on clients (Only Fed) leads to poor performance, primarily due to distribution differences between the source domain and target client domain data. Global models trained via FL typically only grasp common and low-level characteristics from clients, often missing out on fine-grained, specific features unique to each client. Consequently, it becomes essential for clients to engage in transfer learning post-obtaining the global model to attain customized client models. The experiment result also indicates that only applying transfer learning without FL (Only TL) can effectively handle distribution shifts to significantly enhance the performance of detection on both LH and HH datasets. However, FedITD combining FL and

deep transfer learning achieves the best detection performance whether in terms of Macro Average F1 or accuracy on both LH and HH datasets. In comparison, although full tuning and PETuning achieve a good performance in the local test after 20 epochs of tuning, they still do not attain a satisfactory performance in the global test. FedITD achieves the highest Macro Average F1 score, surpassing the Full-tuning method by 2.62% and 27.08% in the local test and global test on the HH dataset respectively. Additionally, FedITD also achieves the best classification accuracy, outperforming the Full-tuning method by 2.64% and 7.21% in the local test and global test on the HH dataset respectively. It successfully detects all five anomaly scenarios in CERT r6.2, whether scenario 2 which has 252 malicious actions and last for 8 weeks, or scenario 5 which only has 4 malicious actions and last for less than 2 minutes.

Based on the information provided by Table 4.8, it is evident that both federated learning and transfer learning play significant roles in the performance of FedITD. Federated learning enables the central server to leverage a broader spectrum of information from multiple clients indirectly, leading to a more generalized global model. On the other hand, transfer learning empowers clients to further improve the global model to adapt to the client's local data, thereby obtaining a more customized client model. On the contrary, due to the diversity of datasets, only federated learning models are more prone to experiencing overfit rising false positive rates or declining sensitivity. Similarly, only transfer learning models may struggle to detect unknown attacks that are not present in the source domain, as it is hard to get knowledge about these attacks from other clients. FedITD can address these shortcomings by constructing a custom powerful model for individual clients using both transfer learning and federated learning. Each client's model is refined to capture their unique behavior traits and detect unknown attacks that do not occur in the local dataset, leading to a more precise, tailored, and general model.

2) **Model Generalization Evaluation**

To demonstrate the superior generalization ability of FedITD, we conducted data processing before the experiment described in the previous subsection to validate its capacity to not only detect insider attacks present in a client's local dataset but also identify unknown attacks. Initially, we removed some specific users' anomaly behavior data from the client's local training and local validation datasets while retaining them in the global test set. Specifically, we excluded user CDE1846's (scenario 4) and user MBG3183's (scenario 5) anomaly instances from client 1's target domain training and local validation datasets while leaving the source domain's datasets unaffected and also keeping them in the global test dataset. Subsequently, we conducted the aforementioned experiment outlined in the previous subsection. Notably, since user CDE1846 and user MBG3183 are absent from client 1's local dataset, models solely trained on local data without any federated learning or transfer learning process struggled to identify user CDE1846 and user MBG3183's anomaly behavior, resulting in poor detection performance in the global test. Conversely, Only TF method, having access to the source domain dataset, could effectively learn user CDE1846 and user MBG3183's anomaly behavior, thus exhibiting superior performance whose Macro Average F1 is 0.9423 and accuracy attains 99.17% in the global test. Despite client 1's limited local data, it successfully detected user CDE1846's anomaly behavior, which had not previously appeared in client 1's local data. In contrast, as Table 4.8 shows, FedITD outperformed OnlyTF in terms of both macro average F1 (0.9517) and accuracy (99.54%). This improvement is attributed to FedITD's excellent ability to facilitate client 1 to learn the knowledge of user CDE1846 and user MBG3183's anomaly behavior from other clients through not only transfer learning but also federated learning. These robust results demonstrated the strong generalization ability of FedITD.

3) **Model Localization Analysis**

Let's dive into transfer learning using the Deep CORAL method: As shown in Figure 4.12, compared with the TL training and local test performance without

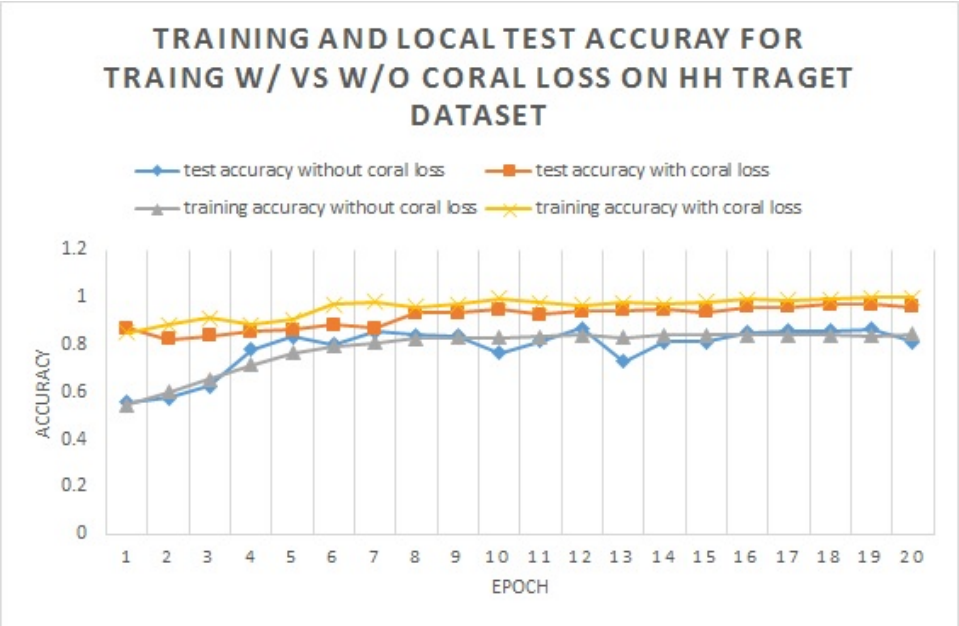

Figure 4.12: Training and Local Test Accuracy for Training w/ vs w/o CORAL Loss on HH Datasets

CORALL Loss, it's evident that incorporating the CORAL loss significantly enhances the performance in the target domain while preserving robust classification accuracy in the source domain. Fine-tuning without domain adaptation may lead to overfitting the model to the source domain, potentially diminishing its performance in the target domain. Integrating the CORAL Loss into the system's loss mitigates the dissimilarities of data distribution between the source and target domains during the fine-tuning process. It helps maintain a balance, ensuring the model performs effectively in both source and target domains.

In Figure 4.13, both the classification loss and the CORAL loss are depicted for training with the CORAL Loss when λ is equal to 0.75 and the learning rate is 0.01. Initially, the CORAL loss is minimal, contrasting with the considerable classification loss. However, after 20 epochs of training, these two losses converge to a similar magnitude.

Next, this study will explore how to set lambda values to optimize the performance

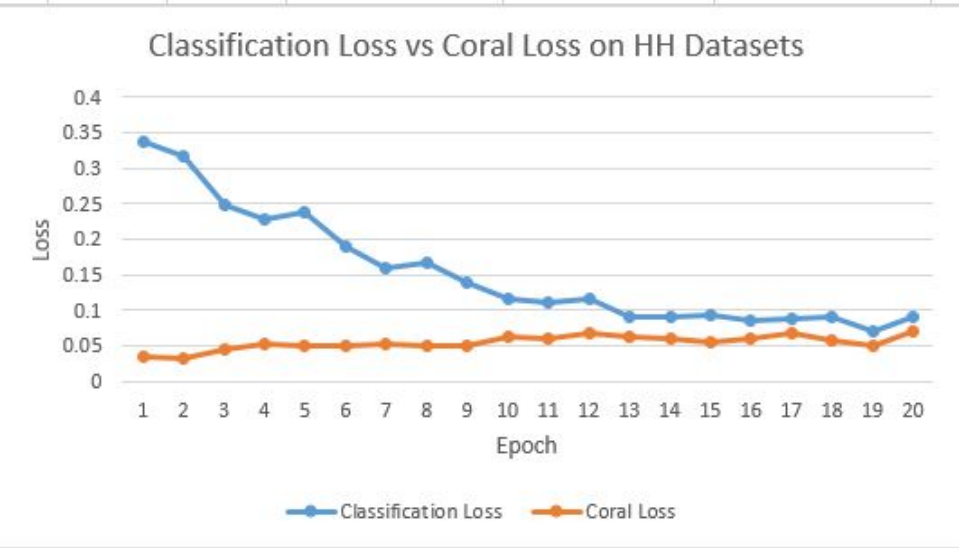

Figure 4.13: Classification Loss vs CORAL Loss on HH Datasets

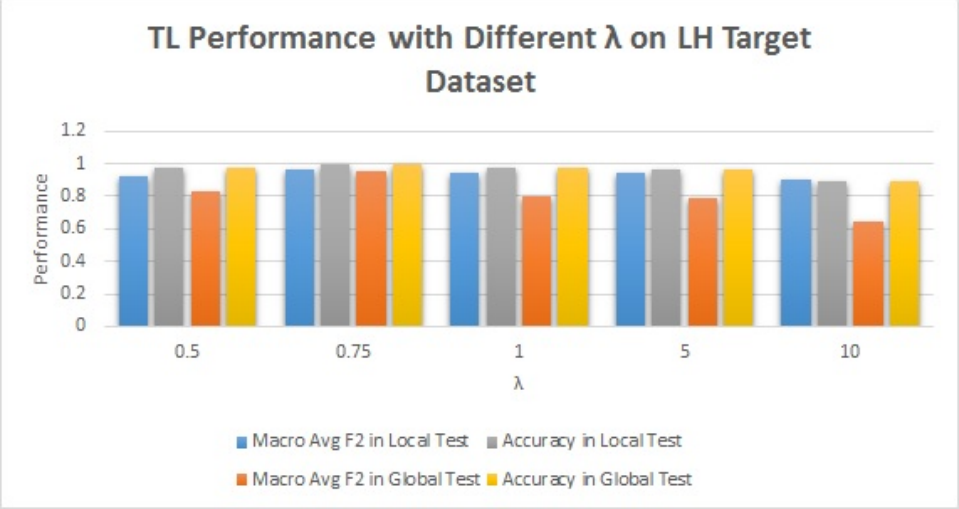

Figure 4.14: TL Performance with different λ with LH Target

on both local test and global test. When the target dataset is the LH dataset, as Figure 4.14 shows, in the local test, with the increasing of λ value, the macro average F1 score increases till it attains the maximum value at 0.9640 and the accuracy also rises till it achieves the top value at 96.43% when λ is equal to 0.75. After that, both macro average F1 score and accuracy decline with the increase of λ. The global test shows the same tendency: with the increase of λ value, the macro average F1 score increases till it attains the maximum value at 0.9517 and the accuracy also rises till it achieves the top value at 99.54% when λ is equal to 0.75.

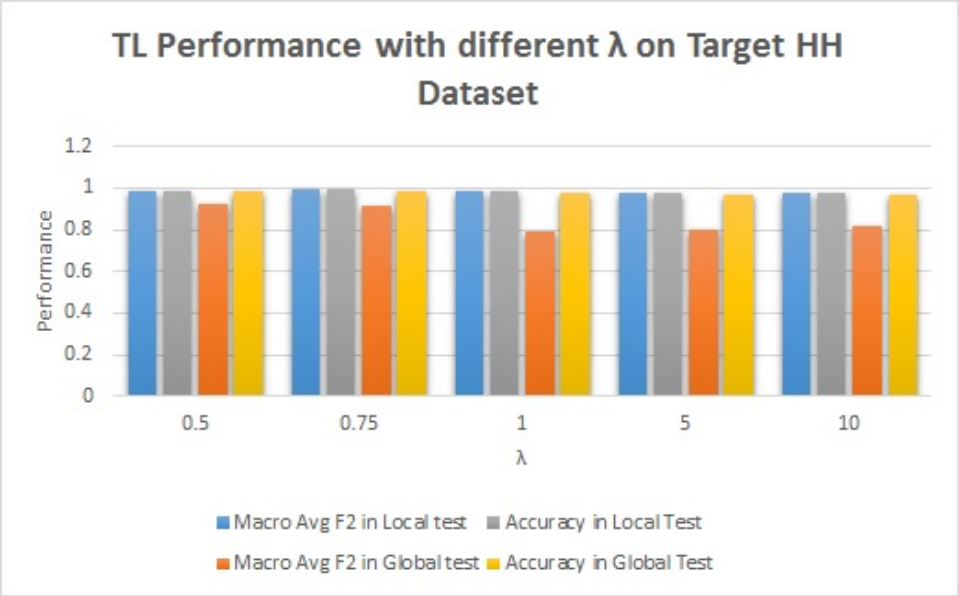

Figure 4.15: TL Performance with different λ with HH Target

When the target domain data is the HH dataset, the case becomes different. As Figure 4.15 shows, when λ is equal to 0.75, the macro average F1 attains the maximum value at 0.9954 and accuracy achieves the top value at 99.54% as well in the local test. However, in the global test, when λ is 0.5, the macro average F1 attains the maximum value at 0.9240 and accuracy achieves the top value at 98.90%.

4.3.2.2 Comparative Study

XLNet with BitFit tuning method that performed the worst in the federated PETuning and then is improved by transfer learning is selected as the representative of FedITD to compare with other methods of insider threat detection. Only one insider threat detection solution based on federated learning is found to compare. This solution uses AutoEncoder to perform the detection [77]. This study also compares insider threat detection methods based on central training on CERT R6.2 dataset. These methods include traditional machine learning algorithms, such as Isolation Forest [60] and one-class SVM [33], and recent deep learning models, such as DistilledTrans [12], LSTM-RNN [65], DeepMIT [67], DD-GCN [43], multistate LSTM + CNN [66], Hierarchical LSTMs [38], simultaneous neural learning [78], unsupervised ensembles [79], log2vec [68], log2vec++ [68], and peering group metadata-informed LSTM Ensembles [80].

The comparative study results show that FedITD not only outperforms Federated AutoEncoder but also defeats all insider threat detection methods based on central training except DistilledTrans [12]. However, the gap between FedITD and DistilledTrans is very narrow: only 1.82% lower in AUC and 1.38% lower in F1 but 4.48% higher in recall. Our proposed FedITD can indirectly combine data from other clients and reduce the discrepancy of data distribution between source and target domain, therefore the anomaly detection rate is even higher than DistiledTrans which utilizes an optimized transform variant model and possesses the advantage of centrally accessing all data samples. Compared to almost all other detection methods whether central trained or federated learned, our method has a great improvement on all performance metrics. It is worth noting that other methods often filter or manipulate the dataset or the model setting to achieve satisfying results. For instance, Unsupervised Ensemble [79] and Federated AutoEncoder [77] require human efforts to examine the highest ranked data to verify if the detection is true positive. The security analysts have to investigate 20% of data instances post-training to get their claimed performance. Additionally, different thresholds set by humans can generate different performance results. For example, log2vec [68] achieves an AUC of

Table 4.9: The Performance Comparison of Various Models

Category	**Method**	**Accuracy**	**Precision**	**Recall**	**F1**	**AUC**
Federated Learning	FedITD	99.54%	92.81%	97.81%	95.17%	97.81%
	Federated AutoEncoder	X	X	97.00%	X	93%
Central Learning	DistilledTrans	99.82%	100.00%	93.33%	96.55%	99.63%
	LSTM-RNN	93.85%	95.12%	92.46%	X	X
	DeepMIT	X	91.60%	93.20%	93.20%	X
	DD-GCN	98.65%	X	X	85.69%	X
	Multistage LSTM +CNN	85.00%	X	X	X	90.47%
	Hierararchical LSTM	X	X	X	X	94.96%
	Simultaneous Neural Learning	X	X	X	X	95.60%
	Unsupervised Ensembles	X	X	X	X	97.70%
	Log2vec	X	X	X	X	86.00%
	Log2vec++	X	X	X	X	93.00%
	Peer Group Metadata-Informed LSTM Ensembles	X	X	X	X	81.40%
	LSTM	X	X	X	X	71.00%
	One Class SVM	X	X	X	X	73.67%
	Isolated Forest	X	X	X	X	76.19%

0.93 by evaluating only 6 potential malicious users and 12 normal users from CERT R6.2 dataset. In comparison, our proposed method is a complete end-to-end solution without any human intervention or manipulation. Additionally, FedITD has no such limitations on the number of users and our higher AUC is achieved by training and testing using the complete dataset. Another benefit of FedITD is the capability to add new clients automatically, eliminating the need for rebuilding the model and incremental algorithms. In the meantime, FedITD also has strong detection performance on client local data that other methods do not have. In brief, FedITD proves its effectiveness in identifying insider anomalies.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis, we propose two innovative solutions based on central learning and federated learning respectively to improve insider threat detection. The proposed DTITD framework is a novel intelligent insider threat detection framework based on digital twins and central trained transformer models. This solution is not only able to monitor the insider risk profile of an organization in a timely and continuous manner but also easily perform in-depth analysis to find root causes and quickly take appropriate remediation actions against insider threats. We then perform novel language data augmentation approaches, including contextual word embedding insert and substitution predicted by the BERT model and context embedding sentence predicted by the GPT-2 model, to overcome the high data imbalance of the sporadic dataset. Our experiment demonstrates that the context embedding sentences predicted by the GPT-2 model outperform the contextual word embedding predicted by the BERT model. Next, this thesis presents a custom transformer model named DistilledTrans and conducts extensive experiments to compare this model with the original transformer model, pre-trained transformer model (BERT+Final layer, RoBERTa + Final layer), hybrid models (BERT + CNN, BERT + LSTM), and the state-of-the-art models. Our experimental results show that:

1. When training models with the sporadic dataset, DistilledTrans trained with the dataset augmented by contextual embedding sentences performs the best in terms of all evaluation metrics, including accuracy, precision, recall, F1-score, and AUC. In addition, this model is lightweight and can significantly reduce training time and costs.

2. When training models with the dense dataset, pre-trained models BERT plus a final layer or RoBERTa plus a final layer can achieve significantly higher performance than all current models, including various hybrid models and two transformer models, with very little sacrifice of precision. Additionally, these models are much more concise and simpler than hybrid models.

On the other hand, we present FedITD, a pioneering framework for insider threat detection that combines FL, PETuning with pre-trained LLMs, and transfer learning. FedITD enables indirect integration of information across clients through FL. To reduce resource costs, minimize time delays, and protect privacy, this study explores three representative PETuning methods—Adapter, BitFit, and LoRA—applied to four pre-trained large language models: BERT, RoBERTa, XLNet, and DistilBERT. The results show that Federated PETuning methods maintain satisfactory performance while effectively lowering resource usage and protecting against privacy breaches. Subsequently, we construct custom client models through transfer learning unsupervised domain adaptation method. Experimental results reveal that FedITD framework addresses the existing issues effectively:

1. **Performance**: FedITD's detection performance surpasses that of other federated methods and performs better than nearly all centrally trained methods, closely approaching the best central training method DistilledTrans.

2. **Efficiency**: It significantly reduces communication costs, storage overhead, memory usage, and time delays.

3. **Adaptability and Generalization**: It demonstrates excellent adaptability to both slightly and highly heterogeneous data and has a strong ability to generalize, detecting unknown attacks that have not appeared in the client's local data.

4. **Optimal Equilibrium**: By tuning system parameters, FedITD can achieve an optimal balance in TL performance across source domain data, target domain data (unlabelled local data), and global testing.

FedITD and DTITD framework compose a complete, highly accurate, and timely insider risk solution at the enterprise level.

5.2 Limitations

First, model training is still considered offline learning, as all models are trained at once on a fixed dataset. Online learning methods, on the other hand, should be used to learn incrementally, continuously updating the models as new data arrives. This allows models to adapt to new trends or changes in data distribution. This is especially important for detecting insider threats in dynamic environments because it enables real-time adaptation, faster updates, and better resource efficiency and scalability.

Second, the models require large datasets for training, as stable and reliable performance cannot be guaranteed without them.

Third, the rich data available in the CERT dataset, including email contents, web browsing data, and user profile information, has not been fully utilized to provide early alerts.

5.3 Future Work

So far, we have proposed and implemented the insider threat framework DTITD and FedITD. However, response time needs further improvement to detect anomalies earlier.

Considering that profound email contents, web browsing contents, and user profile information e.g., psychological metrics, in the CERT dataset have not been utilized yet, our future work would be to perform sentimental analytics on the contents and take advantage of user profile data to provide early alarms. Seq2Seq approach can be used to convert the analysis of aspect categories and sentiment labels to sentence pair classification. One challenge lies in that the email content in the CERT dataset is not natural language but some random combination of unrelated words. This makes us lack useful training data. By systematically integrating human feedback into the training of the email generation model using Reinforcement Learning from Human Feedback (RLHF), an AI system capable of producing more human-like and contextually appropriate email content tailored to specific scenarios and contexts can be created. In addition, leveraging our system to train data online could further enhance the system's power by enabling continuous unsupervised domain adaptation and learning from new observations in the future.

References

[1] J. Care, P. Furtado, and B. Predovich. "Market guide for insider risk management solutions." (2020), [Online]. Available: https://www.gartner.com/document/3994931?ref=solrAll&refval=363319695. (accessed: 03.05.2023).

[2] D. Systems. "Insider risk report." (), [Online]. Available: https://www2.dtexsystems.com/2022-insider-risk-report. (accessed: 03.05.2023).

[3] P. Institute. "2022 ponemon cost of insider threats global report." (2022), [Online]. Available: https://www.proofpoint.com/us/resources/threat-reports/cost-of-insider-threats. (accessed: 05.25.2024).

[4] S. Jose, "Insider threat report 2019," *CA Technol.*, 2019.

[5] CSO, U. S. S. C. D. of SRI-CMU, and ForcePoint, "2018 u.s. state of cybercrime," *Tech. Rep.*, 2018.

[6] J. Care, P. Furtado, and B. Predovich. "Market guide for insider risk management solutions." (2022), [Online]. Available: https://www.gartner.com/document/4013691?ref=solrAll&refval=363320574. (accessed: 03.05.2023).

[7] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[10] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[11] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[12] Z. Q. Wang and A. El Saddik, "Dtitd: An intelligent insider threat detection framework based on digital twin and self-attention based deep learning models," *IEEE Access*, 2023.

[13] N. Houlsby, A. Giurgiu, S. Jastrzebski, *et al.*, "Parameter-efficient transfer learning for nlp," in *International conference on machine learning*, PMLR, 2019, pp. 2790–2799.

[14] E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[15] E. B. Zaken, S. Ravfogel, and Y. Goldberg, "Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models," *arXiv preprint arXiv:2106.10199*, 2021.

[16] Z. Zhang, Y. Yang, Y. Dai, *et al.*, "Fedpetuning: When federated learning meets the parameter-efficient tuning methods of pre-trained language models," in *Annual Meeting of the Association of Computational Linguistics 2023*, Association for Computational Linguistics (ACL), 2023, pp. 9963–9977.

[17] N. Nguyen, P. Reiher, and G. H. Kuenning, "Detecting insider threats by monitoring system call activity," in *IEEE Systems, Man and Cybernetics SocietyInformation Assurance Workshop, 2003.*, IEEE, 2003, pp. 45–52.

[18] M. Hanley and J. Montelibano, "Insider threat control: Using centralized logging to detect data exfiltration near insider termination," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, Tech. Rep., 2011.

[19] M. A. Maloof and G. D. Stephens, "Elicit: A system for detecting insiders who violate need-to-know," in *Recent Advances in Intrusion Detection: 10th International Symposium, RAID 2007, Gold Goast, Australia, September 5-7, 2007. Proceedings 10*, Springer, 2007, pp. 146–166.

[20] R. A. Maxion and T. N. Townsend, "Masquerade detection using truncated command lines," in *Proceedings international conference on dependable systems and networks*, IEEE, 2002, pp. 219–228.

[21] M. B. Salem and S. J. Stolfo, "Detecting masqueraders: A comparison of one-class bag-of-words user behavior modeling techniques.," *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 1, no. 1, pp. 3–13, 2010.

[22] M. B. Salem and S. J. Stolfo, "A comparison of one-class bag-of-words user behavior modeling techniques for masquerade detection," *Security and Communication Networks*, vol. 5, no. 8, pp. 863–872, 2012.

[23] P. Kudłacik, P. Porwik, and T. Wesołowski, "Fuzzy approach for intrusion detection based on user's commands," *Soft Computing*, vol. 20, pp. 2705–2719, 2016.

[24] Y. Song, M. B. Salem, S. Hershkop, and S. J. Stolfo, "System level user behavior biometrics using fisher features and gaussian mixture models," in *2013 IEEE Security and Privacy Workshops*, IEEE, 2013, pp. 52–59.

[25] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale netflow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 129–138.

[26] W. T. Young, H. G. Goldberg, A. Memory, J. F. Sartain, and T. E. Senator, "Use of domain knowledge to detect insider threats in computer activities," in *2013 IEEE Security and Privacy Workshops*, IEEE, 2013, pp. 60–67.

[27] T. E. Senator, H. G. Goldberg, A. Memory, *et al.*, "Detecting insider threats in a real corporate database of computer usage activity," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 1393–1401.

[28] T.-F. Yen, A. Oprea, K. Onarlioglu, *et al.*, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proceedings of the 29th annual computer security applications conference*, 2013, pp. 199–208.

[29] A. Gamachchi, L. Sun, and S. Boztas, "A graph based framework for malicious insider threat detection," *arXiv preprint arXiv:1809.00141*, 2018.

[30] D. C. Le, N. Zincir-Heywood, and M. I. Heywood, "Analyzing data granularity levels for insider threat detection using machine learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 30–44, 2020.

[31] T. Al-Shehari and R. A. Alsowail, "Random resampling algorithms for addressing the imbalanced dataset classes in insider threat detection," *International Journal of Information Security*, vol. 22, no. 3, pp. 611–629, 2023.

[32] O. Brdiczka, J. Liu, B. Price, *et al.*, "Proactive insider threat detection through graph learning and psychological context," in *2012 IEEE Symposium on Security and Privacy Workshops*, IEEE, 2012, pp. 142–149.

[33] L. Lin, S. Zhong, C. Jia, and K. Chen, "Insider threat detection based on deep belief network feature representation," in *2017 International Conference on Green Informatics (ICGI)*, IEEE, 2017, pp. 54–59.

[34] L. Liu, O. De Vel, C. Chen, J. Zhang, and Y. Xiang, "Anomaly-based insider threat detection using deep autoencoders," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, IEEE, 2018, pp. 39–48.

[35] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," *arXiv preprint arXiv:1710.00811*, 2017.

[36] J. Lu and R. K. Wong, "Insider threat detection with long short-term memory," in *Proceedings of the Australasian Computer Science Week Multiconference*, 2019, pp. 1–10.

[37] F. Yuan, Y. Cao, Y. Shang, Y. Liu, J. Tan, and B. Fang, "Insider threat detection with deep neural network," in *Computational Science–ICCS 2018: 18th International Conference, Wuxi, China, June 11–13, 2018, Proceedings, Part I 18*, Springer, 2018, pp. 43–54.

[38] S. Yuan, P. Zheng, X. Wu, and Q. Li, "Insider threat detection via hierarchical neural temporal point processes," in *2019 IEEE International Conference on Big Data (Big Data)*, IEEE, 2019, pp. 1343–1350.

[39] M. A. Haq, M. A. R. Khan, and M. Alshehri, "Insider threat detection based on nlp word embedding and machine learning," *Intell. Autom. Soft Comput*, vol. 33, no. 1, pp. 619–635, 2022.

[40] M. Singh, B. M. Mehtre, S. Sangeetha, and V. Govindaraju, "User behaviour based insider threat detection using a hybrid learning approach," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 4, pp. 4573–4593, 2023.

[41] A. Saaudi, Z. Al-Ibadi, Y. Tong, and C. Farkas, "Insider threats detection using cnn-lstm model," in *2018 International conference on computational science and computational intelligence (CSCI)*, IEEE, 2018, pp. 94–99.

[42] J. Jiang, J. Chen, T. Gu, *et al.*, "Anomaly detection with graph convolutional networks for insider threat and fraud detection," in *MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM)*, IEEE, 2019, pp. 109–114.

[43] X. Li, X. Li, J. Jia, *et al.*, "A high accuracy and adaptive anomaly detection model with dual-domain graph convolutional network for insider threat detection," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1638–1652, 2023.

[44] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[45] Z. Wu, H. Zhang, P. Wang, and Z. Sun, "Rtids: A robust transformer-based approach for intrusion detection system," *IEEE Access*, vol. 10, pp. 64 375–64 387, 2022.

[46] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.* "Improving language understanding by generative pre-training." (2018), [Online]. Available: `https://www.mikecaptain.com/resources/pdf/GPT-1.pdf`. (accessed: 04.05.2025).

[47] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," *Transdisciplinary perspectives on complex systems: New findings and approaches*, pp. 85–113, 2017.

[48] A. El Saddik, "Digital twins: The convergence of multimedia technologies," *IEEE multimedia*, vol. 25, no. 2, pp. 87–92, 2018.

[49] L. Qu, Y. Zhou, P. P. Liang, *et al.*, "Rethinking architecture design for tackling data heterogeneity in federated learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 061–10 071.

[50] O. Weller, M. Marone, V. Braverman, D. Lawrie, and B. Van Durme, "Pretrained models for multilingual federated learning," *arXiv preprint arXiv:2206.02291*, 2022.

[51] G. Sun, M. Mendieta, T. Yang, and C. Chen. "Exploring parameter-efficient fine-tuning for improving communication efficiency in federated learning." (2022), [Online]. Available: `https://openreview.net/pdf?id=EmH1WE1fRbt`. (accessed: 04.05.2025).

[52] J. Chen, W. Xu, S. Guo, J. Wang, J. Zhang, and H. Wang, "Fedtune: A deep dive into efficient federated fine-tuning with pre-trained transformers," *arXiv preprint arXiv:2211.08025*, 2022.

[53] Y. Chen, X. Qin, J. Wang, C. Yu, and W. Gao, "Fedhealth: A federated transfer learning framework for wearable healthcare," *IEEE Intelligent Systems*, vol. 35, no. 4, pp. 83–93, 2020.

[54] Y. Fan, Y. Li, M. Zhan, H. Cui, and Y. Zhang, "Iotdefender: A federated transfer learning intrusion detection framework for 5g iot," in *2020 IEEE 14th international conference on big data science and engineering (BigDataSE)*, IEEE, 2020, pp. 88–95.

[55] B. Sun, J. Feng, and K. Saenko, "Return of frustratingly easy domain adaptation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[56] B. Sun and K. Saenko, "Deep coral: Correlation alignment for deep domain adaptation," in *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*, Springer, 2016, pp. 443–450.

[57] S. E. Institute, *Insider threat test dataset*, `https://doi.org/10.1184/R1/12841247.v1`, Accessed: 2025-04-10, 2016.

[58] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[59] T. Rashid, I. Agrafiotis, and J. R. Nurse, "A new take on detecting insider threats: Exploring the use of hidden markov models," in *Proceedings of the 8th ACM CCS International workshop on managing insider security threats*, 2016, pp. 47–56.

[60] B. Lv, D. Wang, Y. Wang, Q. Lv, and D. Lu, "A hybrid model based on multi-dimensional features for insider threat detection," in *Wireless Algorithms, Systems, and Applications: 13th International Conference, WASA 2018, Tianjin, China, June 20-22, 2018, Proceedings 13*, Springer, 2018, pp. 333–344.

[61] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1285–1298.

[62] R. Nasir, M. Afzal, R. Latif, and W. Iqbal, "Behavioral based insider threat detection using deep learning," *IEEE Access*, vol. 9, pp. 143 266–143 274, 2021.

[63] B. Sharma, P. Pokharel, and B. Joshi, "User behavior analytics for anomaly detection using lstm autoencoder-insider threat detection," in *Proceedings of the 11th International Conference on Advances in Information Technology*, 2020, pp. 1–9.

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

[65] F. Meng, F. Lou, Y. Fu, and Z. Tian, "Deep learning based attribute classification insider threat detection for data security," in *2018 IEEE third international conference on data science in cyberspace (DSC)*, IEEE, 2018, pp. 576–581.

[66] M. Singh, B. M. Mehtre, and S. Sangeetha, "User behavior profiling using ensemble approach for insider threat detection," in *2019 IEEE 5th International Conference on Identity, Security, and Behavior Analysis (ISBA)*, IEEE, 2019, pp. 1–8.

[67] D. Sun, M. Liu, M. Li, Z. Shi, P. Liu, and X. Wang, "Deepmit: A novel malicious insider threat detection framework based on recurrent neural network," in *2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, IEEE, 2021, pp. 335–341.

[68] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 1777–1794.

[69] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[70] H. Liu, S. Zhang, P. Zhang, *et al.*, "Blockchain and federated learning for collaborative intrusion detection in vehicular edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 6, pp. 6073–6084, 2021.

[71] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*, Springer, 1999, pp. 223–238.

[72] L. Muñoz-González, K. T. Co, and E. C. Lupu, "Byzantine-robust federated machine learning through adaptive model averaging," *arXiv preprint arXiv:1909.05125*, 2019.

[73] P.-C. Cheng, K. Eykholt, Z. Gu, *et al.*, "Separation of powers in federated learning (poster paper)," in *Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning*, 2021, pp. 16–18.

[74] S. Hu, N. Ding, W. Zhao, *et al.*, "Opendelta: A plug-and-play library for parameter-efficient adaptation of pre-trained models," *arXiv preprint arXiv:2307.03084*, 2023.

[75] D. Zeng, S. Liang, X. Hu, H. Wang, and Z. Xu, "Fedlab: A flexible federated learning framework," *Journal of Machine Learning Research*, vol. 24, no. 100, pp. 1–7, 2023. [Online]. Available: `http://jmlr.org/papers/v24/22-0440.html`.

[76] L. Zhu, Z. Liu, and S. Han, *Deep leakage from gradients*, 2019. arXiv: `1906.08935 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1906.08935`.

[77] M. Amiri-Zarandi, H. Karimipour, and R. A. Dara, "A federated and explainable approach for insider threat detection in iot," *Internet of Things*, vol. 24, p. 100 965, 2023.

[78] L. Liu, C. Chen, J. Zhang, O. De Vel, and Y. Xiang, "Insider threat identification using the simultaneous neural learning of multi-source logs," *IEEE Access*, vol. 7, pp. 183 162–183 176, 2019.

[79] D. C. Le and N. Zincir-Heywood, "Anomaly detection for insider threats using unsupervised ensembles," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1152–1164, 2021.

[80] J. Matterer and D. LeJeune, "Peer group metadata-informed lstm ensembles for insider threat detection," in *The Thirty-First International Flairs Conference*, 2018.