https://doi.org/10.7840/kics.2025.50.10.1631

# Pre-Trained Large Language Model Pipeline for Anomaly Detection Based on the MITRE ATT&CK Framework

Kyuchang Kang◆°, Yu-Jin So*, Jong-Geun Park*

## ABSTRACT

In this paper, we propose a Large Language Model (LLM) pipeline utilizing the UWF-ZeekData22 dataset based on MITRE ATT&CK Matrix to address the growing cyber threats in modern society. We first performed an exploratory data analysis (EDA) to derive key feature groups that reflect the spatio-temporal characteristics and connectivity of network traffic logs. The derived feature groups are used to generate input sequences for pre-training the BERT model. In the pre-training phase, we applied a masked language model (MLM) task to effectively learn network traffic patterns and achieved a mask prediction accuracy of over 0.9. In the fine-tuning and inference phase, we optimized the models for anomaly detection by adopting a weighted sampling technique to handle the imbalance problem of each tactic in the dataset. The performance evaluation showed that all models had an accuracy above 0.94 and an AUC-ROC value close to 1.0. We also analyzed the impact of the padding method according to model size and found that static padding performed better for large models, while dynamic padding performed better for small models. These results demonstrate that LLM-based pre-training can successfully learn complex patterns of network traffic logs and can reliably detect various tactics. Therefore, the proposal of this paper is expected to provide a practical case study in the modernization of network security systems and the development of real-time security monitoring solutions.

**Key Words :** Anomaly Detection, Pre-trained Large Language Model (LLM), MITRE ATT&CK Framework, Network Logs Analysis, Feature Engineering, Cybersecurity

## Ⅰ. Introduction

In modern society, the complexity of network environments is increasing as digitalization is accelerating. In particular, new types of cybersecurity threats are growing rapidly due to the popularization of IoT devices and the increasing network logs. As a result, detecting cybersecurity threats and building an effective response system has become an important challenge.

Previous works has focused primarily on detecting attacks by directly analyzing network logs. To do this, researchers have primarily leveraged IDS datasets such as KDD Cup '99, NSL-KDD, etc. to analyze various network attack scenarios. And they mainly studied anomaly detection models based on the technical characteristics of the attacks.

As cyber threats have become more sophisticated, researchers have recently focused on understanding not only the technical aspects of attacks, but also their strategic planning and intent. As a result, researchers are leveraging frameworks such as MITRE

※ This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MND) (RS-2022-11220701, Development of Security Technology for Interworking between M-BcN and 5G Commercial Network)

◆° First and Corresponding Author : Kunsan National University, IT Convergence & Communication Eng., kc.kang@kunsan.ac.kr, 정회원

* Electronics and Telecommunications Research Institute (ETRI), sy120236@etri.re.kr; queue@etri.re.kr, 정회원

논문번호 : 202503-050-E-RE, Received March 5, 2025; Revised April 16, 2025; Accepted May 7, 2025

ATT&CK[1] to understand the tactical intent of attackers and counter attacks. However, anomaly detection researches based on MITRE ATT&CK have mostly been studied with conventional machine learning or graph-based approaches, rather than LLMs.

LLMs can effectively analyze unstructured data by leveraging their natural language processing (NLP) capabilities. Since LLMs can learn complex patterns, they can detect new threats such as zero-day attacks well.

In this paper, we propose a pipeline of tactical information-based anomaly detection models to counter cybersecurity threats. We also aim to validate the applicability of LLM pre-training models for tactical information-based anomaly detection.

For an experimental dataset, we used UWF-ZeekData22[2], a network log dataset that reflects a modern network environment based on the MITRE ATT&CK matrix. We considered this dataset to be suitable as an experimental dataset because it contains information about attackers' tactics and modern attack patterns.

In this paper, we applied the pre-training mechanism of BERT model to the field of network security to improve the ability to recognize complex network log patterns. Unlike traditional machine learning approaches or simple statistical-based models, we aim to greatly improve the efficiency of sequence pattern learning by utilizing the context-awareness of the transformer architecture, which has been proven in the field of natural language processing, for network log analysis. In addition, we apply log-scale-based weighted sampling and low-frequency tactical preservation strategies to solve the imbalanced data problem that occurs in real-world network environments. By doing so, we aim to improve the detection rate of sparse but important attack patterns in real-world security environments and overcome the limitations of previous imbalanced data processing methodologies.

In addition, we empirically analyze the padding strategy for efficient implementation of LLM-based models in network security environments. By systematically analyzing the effectiveness of static and dynamic padding techniques for BERT models of different sizes, we aim to provide practical guidelines for efficient model deployment in real-time security monitoring systems and large-scale network environments. Through this study, we hope to expand the applicability of deep learning technologies, especially LLM-based supervised learning models, in the field of network security and provide practical solutions to various challenges in real-world security environments.

The remainder of this paper is organized as follows. Section 2 describes related work on network intrusion detection datasets, LLMs, and tactical data utilization. Section 3 describes in detail the implementation process of the proposed LLM-based pre-training model pipeline. In section 4, we validate the network anomaly detection performance of the proposed model through various experiments and evaluation metrics. Finally, section 5 summarizes the paper and discusses future research directions.

## Ⅱ. Related Works

### 2.1 Network Intrusion Detection Dataset

High-quality datasets that reflect real-world network environments are a critical part of developing and evaluating network intrusion detection systems (IDS). In particular, there is a growing need for up-to-date network log data that reflects a variety of protocols and different types of attacks. Datasets that reflect the latest attack techniques and real-world network environments help develop effective anomaly detection models. However, traditional IDS datasets are limited in their ability to detect the complexity of modern networks and evolving attack techniques.

KDD Cup 99[3,4] is a TCP/IP-based network log dataset developed by the DARPA IDS evaluation program in 1999. It contains 41 features and 22 attack types such as DoS, Probe, U2R, R2L, etc. However, it was generated in an old network environment and contained duplicate records, which can skew the learning results.

NSL-KDD[4] is a dataset proposed to address some of the issues in KDD Cup 99, particularly the duplicate records and class imbalance problems. However, this dataset still does not reflect the latest network protocols and the latest attack patterns.

UNSW-NB15[5] is a dataset collected from real-world network environments. It provides 49 features and 9 different attack types such as Generic, Exploits, Fuzzers, DoS, etc. However, this data is biased towards certain attack types, which means it can't address the latest advanced threats, such as AI-driven attacks.

UGR16[6] is a dataset collected from an ISP (Internet Service Provider) network environment in 2016, consisting of four months of normal traffic and two months of attack traffic. This dataset has the advantage of using netflow-based features to analyze traffic patterns over a long period of time, but has the disadvantage of being limited to a specific network environment with a limited number of attack types.

CICIDS2017[7] is a dataset produced by the Canadian Cyber Security Research Institute in 2017. It contains 78 network features and various modern attack scenarios such as Brute Force, DoS/DDoS, and Web Attack, etc. However, the data collection period is only 5 days, which limits its ability to analyze traffic patterns over a long period of time. Also, some attack types have an unbalanced distribution of data.

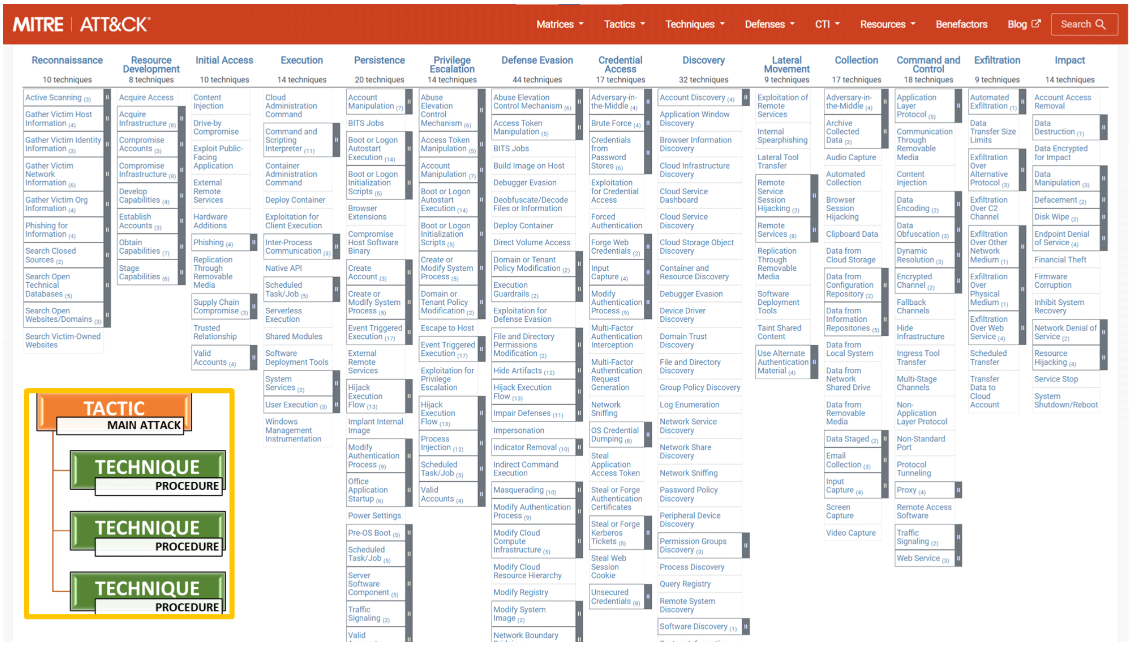
As we have seen above, traditional IDS datasets have been widely used for intrusion detection. However, they have a common limitation that they cannot fully respond to changes in the network environment and new cyberattacks. Therefore, in this paper, we chose the UWF-ZeekData22 dataset based on the MITRE ATT&CK matrix, which reflects the latest network environment.

## 2.2 MITRE ATT&CK

MITRE is a non-profit research organization in the United States that plays a key role in national security and cybersecurity. The organization develops and provides key security frameworks to the global security community, including CVE (Common Vulnerabilities and Exposures) and ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge).

MITRE ATT&CK is a knowledge-based framework that systematically categorizes the malicious behavior of actual cyber attackers according to TTPs. The TTPs consist of attack objectives (Tactics), attack methods (Techniques) to achieve attack objectives, and detailed procedures (Procedures) to execute the attack methods.

Figure 1 is a visualization of the overall structure of the MITRE ATT&CK Matrix for Enterprise. This framework is continually updated to include the latest



Reconnaissance
10 techniques
Active Scanning (3)
Gather Victim Host Information (4)
Gather Victim Identity Information (3)
Gather Victim Network Information (6)
Gather Victim Org Information (4)
Phishing for Information (4)
Search Closed Sources (2)
Search Open Technical Databases (5)
Search Open Websites/Domains (3)
Search Victim-Owned Websites

Resource Development
8 techniques
Acquire Access
Acquire Infrastructure (8)
Compromise Accounts (3)
Compromise Infrastructure (8)
Develop Capabilities (4)
Establish Accounts (3)
Obtain Capabilities (7)
Stage Capabilities (6)

Initial Access
10 techniques
Content Injection
Drive-by Compromise
Exploit Public-Facing Application
External Remote Services
Hardware Additions
Phishing (4)
Replication Through Removable Media
Supply Chain Compromise (3)
Trusted Relationship
Valid Accounts (4)

Execution
14 techniques
Cloud Administration Command
Command and Scripting Interpreter (11)
Container Administration Command
Deploy Container
Exploitation for Client Execution
Inter-Process Communication (3)
Native API
Scheduled Task/Job (5)
Serverless Execution
Shared Modules
Software Deployment Tools
System Services (2)
User Execution (3)
Windows Management Instrumentation

Persistence
20 techniques
Account Manipulation (7)
BITS Jobs
Boot or Logon Autostart Execution (14)
Boot or Logon Initialization Scripts (5)
Browser Extensions
Compromise Host Software Binary
Create Account (3)
Create or Modify System Process (5)
Event Triggered Execution (17)
External Remote Services
Hijack Execution Flow (13)
Implant Internal Image
Modify Authentication Process (9)
Office Application Startup (6)
Power Settings
Pre-OS Boot (5)
Scheduled Task/Job (5)
Server Software Component (5)
Traffic Signaling (2)
Valid

Privilege Escalation
14 techniques
Abuse Elevation Control Mechanism (6)
Access Token Manipulation (5)
Account Manipulation (7)
Boot or Logon Autostart Execution (14)
Boot or Logon Initialization Scripts (5)
Create or Modify System Process (5)
Domain or Tenant Policy Modification (2)
Escape to Host
Event Triggered Execution (17)
Exploitation for Privilege Escalation
Hijack Execution Flow (13)
Process Injection (12)
Scheduled Task/Job (5)
Valid Accounts (4)

Defense Evasion
44 techniques
Abuse Elevation Control Mechanism (6)
Access Token Manipulation (5)
BITS Jobs
Build Image on Host
Debugger Evasion
Deobfuscate/Decode Files or Information
Deploy Container
Direct Volume Access
Domain or Tenant Policy Modification (2)
Execution Guardrails (2)
Exploitation for Defense Evasion
File and Directory Permissions Modification (2)
Hide Artifacts (12)
Hijack Execution Flow (13)
Impair Defenses (11)
Impersonation
Indicator Removal (10)
Indirect Command Execution
Masquerading (10)
Modify Authentication Process (9)
Modify Cloud Compute Infrastructure (5)
Modify Cloud Resource Hierarchy
Modify Registry
Modify System Image (2)
Network Boundary

Credential Access
17 techniques
Adversary-in-the-Middle (4)
Brute Force (4)
Credentials from Password Stores (6)
Exploitation for Credential Access
Forced Authentication
Forge Web Credentials (2)
Input Capture (4)
Modify Authentication Process (9)
Multi-Factor Authentication Interception
Multi-Factor Authentication Request Generation
Network Sniffing
OS Credential Dumping (8)
Steal Application Access Token
Steal or Forge Authentication Certificates
Steal or Forge Kerberos Tickets (5)
Steal Web Session Cookie
Unsecured Credentials (8)

Discovery
32 techniques
Account Discovery (4)
Application Window Discovery
Browser Information Discovery
Cloud Infrastructure Discovery
Cloud Service Dashboard
Cloud Service Discovery
Cloud Storage Object Discovery
Container and Resource Discovery
Debugger Evasion
Device Driver Discovery
Domain Trust Discovery
File and Directory Discovery
Group Policy Discovery
Log Enumeration
Network Service Discovery
Network Share Discovery
Network Sniffing
Password Policy Discovery
Peripheral Device Discovery
Permission Groups Discovery (3)
Process Discovery
Query Registry
Remote System Discovery
Software Discovery (1)

Lateral Movement
9 techniques
Exploitation of Remote Services
Internal Spearphishing
Lateral Tool Transfer
Remote Service Session Hijacking (2)
Remote Services (8)
Replication Through Removable Media
Software Deployment Tools
Taint Shared Content
Use Alternate Authentication Material (4)

Collection
17 techniques
Adversary-in-the-Middle (4)
Archive Collected Data (3)
Audio Capture
Automated Collection
Browser Session Hijacking
Clipboard Data
Data from Cloud Storage
Data from Configuration Repository (2)
Data from Information Repositories (5)
Data from Local System
Data from Network Shared Drive
Data from Removable Media
Data Staged (2)
Email Collection (3)
Input Capture (4)
Screen Capture
Video Capture

Command and Control
18 techniques
Application Layer Protocol (5)
Communication Through Removable Media
Content Injection
Data Encoding (2)
Data Obfuscation (3)
Dynamic Resolution (3)
Encrypted Channel (2)
Fallback Channels
Hide Infrastructure
Ingress Tool Transfer
Multi-Stage Channels
Non-Application Layer Protocol
Non-Standard Port
Protocol Tunneling
Proxy (4)
Remote Access Software
Traffic Signaling (2)
Web Service (3)

Exfiltration
9 techniques
Automated Exfiltration (1)
Data Transfer Size Limits
Exfiltration Over Alternative Protocol (3)
Exfiltration Over C2 Channel
Exfiltration Over Other Network Medium (1)
Exfiltration Over Physical Medium (1)
Exfiltration Over Web Service (4)
Scheduled Transfer
Transfer Data to Cloud Account

Impact
14 techniques
Account Access Removal
Data Destruction (1)
Data Encrypted for Impact
Data Manipulation (3)
Defacement (2)
Disk Wipe (2)
Endpoint Denial of Service (4)
Financial Theft
Firmware Corruption
Inhibit System Recovery
Network Denial of Service (2)
Resource Hijacking (4)
Service Stop
System Shutdown/Reboot

Fig. 1. MITRE ATT&CK Matrix for Enterprise

cyber threats. It is an important tool for security professionals to identify threats and formulate effective response strategies.

MITRE ATT&CK is used in a variety of security domains, including security R&D, threat intelligence, intrusion detection systems (IDS), and incident response (IR). The framework standardizes attack methods to enable detailed analysis of cyber threats. It also assists organizations in planning effective security strategies and improving their response.

MITRE ATT&CK has become a de facto standard and serves as a core reference framework for the development of modern security solutions and threat response strategies.

### 2.3 Large Language Models (LLMs)

LLMs have been proven to be an effective model for learning complex correlations of high-dimensional data in natural language processing[8]. LLMs have the advantage of understanding and analyzing unstructured security data based on their powerful representation learning capabilities[9]. These advantages have been applied to a variety of security tasks such as log analysis, malware detection, and vulnerability identification[10-12].

Previous researchers have proposed using LLMs to automatically extract features and learn patterns from the data. LLMs can transfer general pattern knowledge learned from large-scale data to new tasks through pre-training. In particular, the self-attention mechanism can effectively model complex temporal and spatial relationships between tokens in data. Therefore, LLMs are expected to make it easier to find attack patterns that are difficult to detect with traditional machine learning models.

Most of the previous studies utilizing the tactical information of the MITRE ATT&CK Matrix applied traditional machine learning techniques[13-16], and few have attempted LLM-based network anomaly detection.

BERT[17] is widely used as a base model in the field of representation learning because of its properties of sequential data processing, context understanding, and learning relationships between features. Given these features, we considered it a good choice as a pre-training model to validate the applicability of LLM[18].

In this paper, we propose a pipeline of LLM-based pre-trained models for network anomaly detection using the UWF-ZeekData22 dataset.

## Ⅲ. Method

In this paper, we propose a pipeline of pre-trained models based on BERT as a methodology for network log anomaly detection. The proposed pipeline consists of five main steps: (1) data preparation & analysis, (2) feature engineering & preprocessing, (3) pre-training, (4) fine-tuning, and (5) inference.

Figure 2 shows a schematic diagram of the overall configuration of the proposed BERT-based pre-training model pipeline and the interactions between each phase.

Figure 2 intuitively illustrates the process of (1) analyzing raw data to understand its structural characteristics in the data preparation and analysis phase, (2) extracting key fields and performing preprocessing in the feature engineering and preprocessing phase, (3) initializing the model in the pre-training phase, (4) optimizing the model for a specific task in the fine-tuning phase, and (5) predicting the final outcome in the inference phase.

The following sections describe in detail how each phase is organized and performed.

### 3.1 Data Preparation & Analysis Phase

We systematically analyzed the structure and properties of the UWF-ZeekData22 dataset to derive key features of network log patterns.

In this section, we used exploratory data analysis (EDA) to explore the overall distribution and properties of the dataset and analyzed key fields that are important for anomaly detection.

UWF-ZeekData22 is a dataset of network log data collected by the Zeek tool and labeled according to the MITRE ATT&CK Matrix. This dataset contains a total of 18,562,468 network log records, with a 50:50 ratio of normal to abnormal network logs.

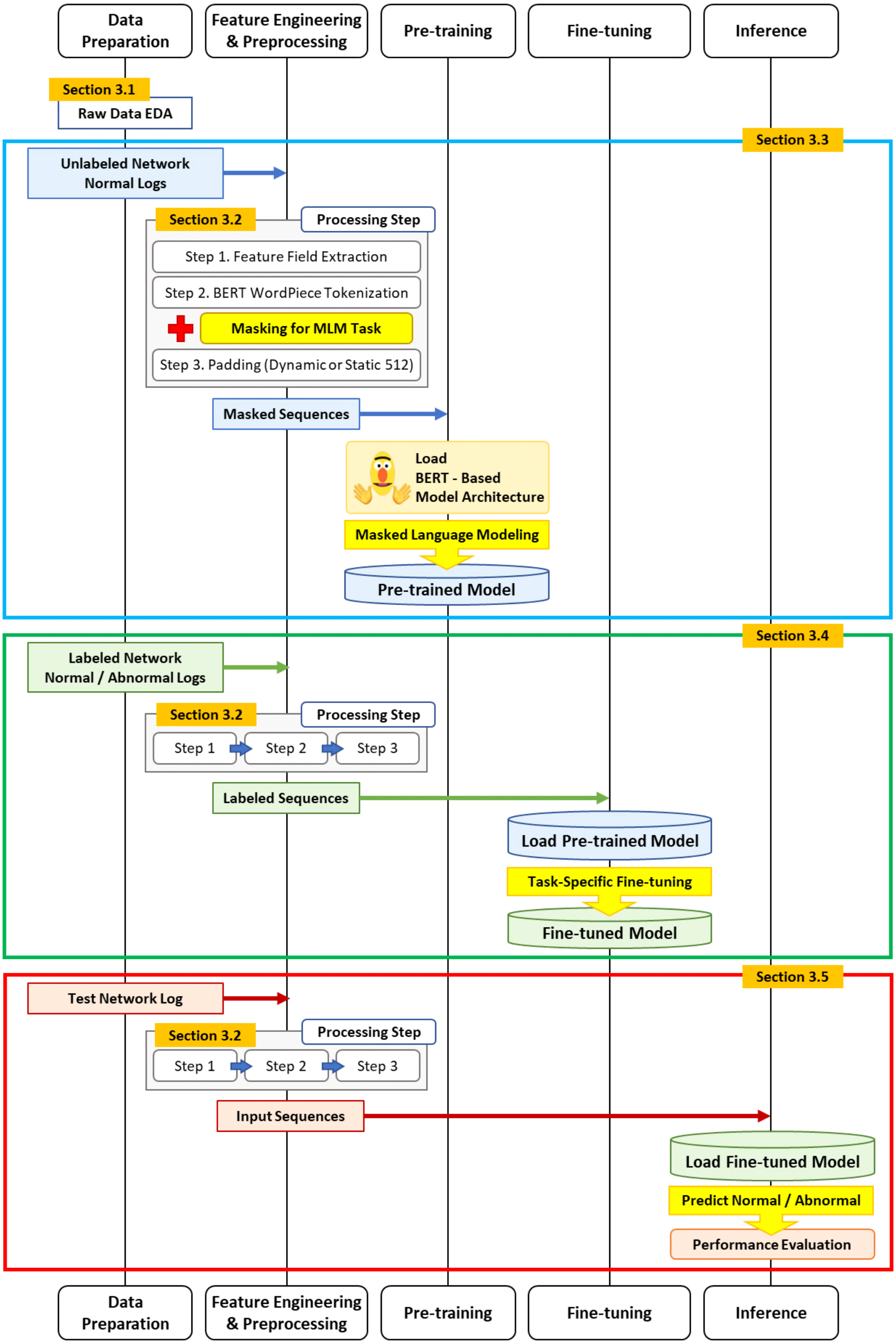


Fig. 2. Structure of a Pre-training Pipeline based on BERT Architecture

We categorized the raw data into three high-level groups to clarify the structural characteristics of the dataset. Figure 3 shows the structure of the dataset according to the classification categories. Table 1 provides a detailed description of the categorization of the dataset.

Through EDA, we identified key features for composing network log patterns, such as network address information (src_ip_zeek, dest_ip_zeek), port information (src_port_zeek, dest_port_zeek), connection

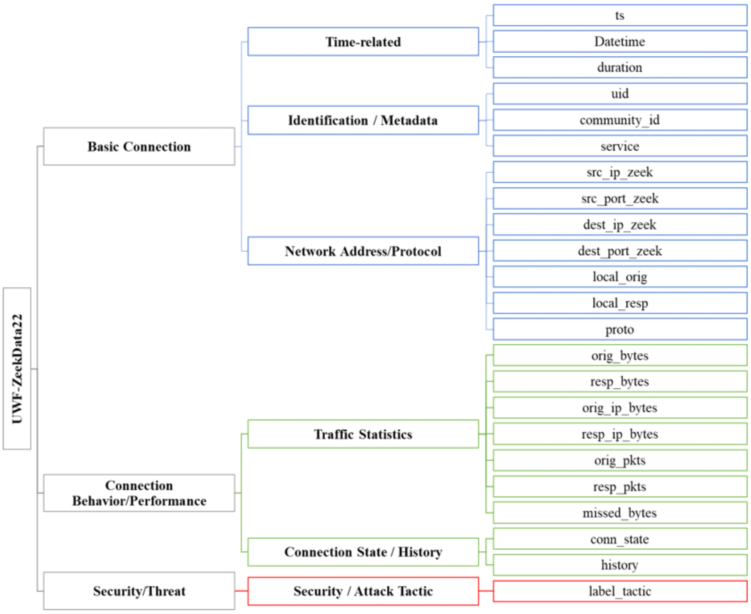


Fig. 3. Overview of UWF-ZeekData22

Table 1. Description of the categorization of the UWF-ZeekData22

| Main Category | Sub-Category | Field | Description |
|---|---|---|---|
| Basic Connection Information | Time-related | ts | Connection start time represented as a Unix timestamp. |
| | | datetime | Human-readable date and time information. |
| | | duration | Duration of the connection in seconds. |
| | Identification/Metadata | uid | Unique identifier assigned to each connection. |
| | | community_id | Hash value used for grouping network flows. |
| | | service | Application layer protocol (e.g., HTTP, DNS, SSH). |
| | Network Address/Protocol | src_ip_zeek | Source IP address. |
| | | src_port_zeek | Source port number. |
| | | dest_ip_zeek | Destination IP address. |
| | | dest_port_zeek | Destination port number. |
| | | local_orig | Indicator of whether the originator is within the local network. |
| | | local_resp | Indicator of whether the responder is within the local network. |
| | | proto | Underlying transport protocol used (e.g., TCP, UDP). |
| Connection Behavior/Performance Information | Traffic Statistics | orig_bytes | Payload byte counts transmitted by the originator. |
| | | resp_bytes | Payload byte counts transmitted by the responder. |
| | | orig_ip_bytes | Total byte counts recorded at the IP layer for the originator. |
| | | resp_ip_bytes | Total byte counts recorded at the IP layer for the responder. |
| | | orig_pkts | Number of packets transmitted by the originator. |
| | | resp_pkts | Number of packets transmitted by the responder. |
| | | missed_bytes | Number of bytes lost during the capture process. |
| | Connection State/History | conn_state | Flag representing the current state of the connection. |
| | | history | Record of TCP flag sequences documenting the connection's state transitions. |
| Security/Threat Information | Security/Attack Tactic | label_tactic | Classification of attack tactics based on the MITRE ATT&CK framework. |

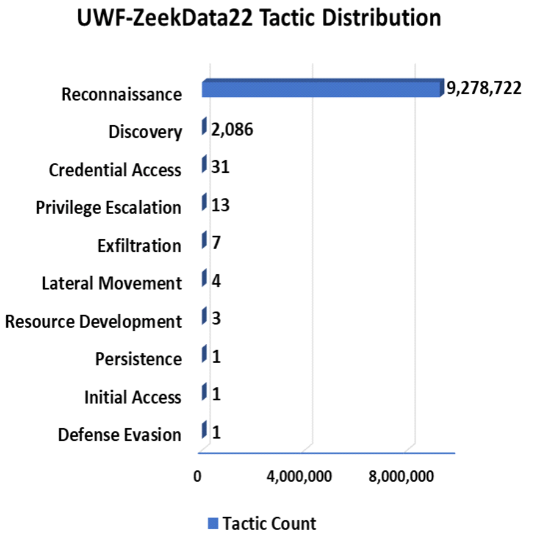


Fig. 4. Tactic Distribution in UWF-ZeekData22

status (conn_state), and traffic volume (orig_bytes, resp_bytes, orig_pkts, resp_pkts).

Figure 4 shows the distribution of the MITRE ATT&CK-based tactics in the dataset. In our analysis, we found that there is a significant imbalance between each tactic.While the Reconnaissance tactic has the highest proportion of occurrences at 9,278,722, other tactics occur much less frequently. For example, there are only 2,086 occurrences of Discovery tactics, 31 occurrences of Credential Access tactics, 13 occurrences of Privilege Escalation tactics, and only 1 occurrence each of Persistence, Initial Access, and Defense Evasion. This imbalance between tactics can lead to a bias toward certain kinds of tactics when training a model. This suggests that we need to adjust the imbalance between classes to ensure a balanced representation of different tactics.

### 3.2 Feature Engineering & Preprocessing Phase

To effectively train network log data with a BERT model, proper preprocessing is required. To this end, we designed a process to convert network traffic data into BERT input sequences. Each processing step consists of Feature Extraction, Tokenization, and Padding as shown in Figure 2.

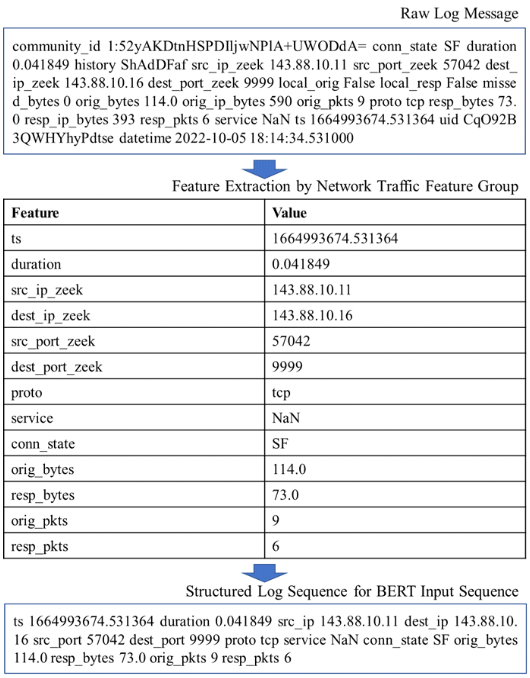
In processing step 1, we categorized the key features for identifying network anomalies derived from the EDA in section 3.1. Based on the inherent characteristics of network logs and the MITRE ATT&CK, we categorized network logs into five feature groups. Table 2 shows the results of the feature grouping.

Figure 5 shows the process of converting raw log messages into sequences. First, we extracted 13 features from the raw log message based on the five categories of Network Traffic Feature Groups. Next, we converted the raw log message into a structured form (Structured Log Sequence) consisting of pairs of the 13 extracted features and their corresponding values. Through this process, we constructed basic data that will serve as the input sequence for processing step 2.

In processing step 2, we performed a tokenization process to convert the feature fields into BERT input sequences. We considered network log data for each field as individual tokens and organized them into a single sentence. Next, we generated a structured BERT input sequence by utilizing the special tokens ([CLS], [SEP], [MASK]) of BERT WordPiece

Table 2. Feature Grouping for Network Traffic

| Feature Group | Feature Fields | Analysis Applications | MITRE ATT&CK Tactics |
|---|---|---|---|
| Temporal Metadata | ts, duration | • Analysis of temporal continuity of attack behaviors<br>• Identification of long-term threats and abnormal sessions | • Persistence<br>• Execution |
| Endpoint Information | src_ip_zeek, dest_ip_zeek, src_port_zeek, dest_port_zeek | • Provides network location information for attack sources and destinations<br>• Analysis of abnormal port usage and scanning behavior patterns | • Initial Access<br>• Discovery |
| Network Traffic Information | orig_bytes, resp_bytes, orig_pkts, resp_pkts | • Analysis of anomaly patterns based on bidirectional traffic volume and packet counts<br>• Detection of large-scale data transfers and resource consumption attacks | • Impact<br>• Collection |
| Service/Protocol Information | proto, service | • Analysis of abnormal protocol usage and service exploitation attempts<br>• Detection of privilege escalation attempts through known vulnerabilities | • Privilege Escalation<br>• Defense Evasion |
| Connection Status | conn_state | • Time-series analysis of connection state changes and failure patterns<br>• Identification of behavioral characteristics in abnormal network access attempts | • Lateral Movement<br>• Command and Control |



Raw Log Message

```
community_id 1:52yAKDtnHSPDIljwNPlA+UWODdA= conn_state SF duration
0.041849 history ShAdDFaf src_ip_zeek 143.88.10.11 src_port_zeek 57042 dest_
ip_zeek 143.88.10.16 dest_port_zeek 9999 local_orig False local_resp False misse
d_bytes 0 orig_bytes 114.0 orig_ip_bytes 590 orig_pkts 9 proto tcp resp_bytes 73.
0 resp_ip_bytes 393 resp_pkts 6 service NaN ts 1664993674.531364 uid CqO92B
3QWHYhyPdtse datetime 2022-10-05 18:14:34.531000
```

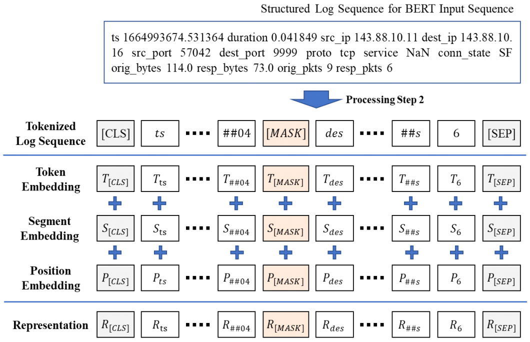
Feature Extraction by Network Traffic Feature Group

| Feature | Value |
|---|---|
| ts | 1664993674.531364 |
| duration | 0.041849 |
| src_ip_zeek | 143.88.10.11 |
| dest_ip_zeek | 143.88.10.16 |
| src_port_zeek | 57042 |
| dest_port_zeek | 9999 |
| proto | tcp |
| service | NaN |
| conn_state | SF |
| orig_bytes | 114.0 |
| resp_bytes | 73.0 |
| orig_pkts | 9 |
| resp_pkts | 6 |

Structured Log Sequence for BERT Input Sequence

```
ts 1664993674.531364 duration 0.041849 src_ip 143.88.10.11 dest_ip 143.88.10.
16 src_port 57042 dest_port 9999 proto tcp service NaN conn_state SF orig_bytes
114.0 resp_bytes 73.0 orig_pkts 9 resp_pkts 6
```

Fig. 5. Example of BERT Input Sequence Conversion Process



Structured Log Sequence for BERT Input Sequence

```
ts 1664993674.531364 duration 0.041849 src_ip 143.88.10.11 dest_ip 143.88.10.
16 src_port 57042 dest_port 9999 proto tcp service NaN conn_state SF
orig_bytes 114.0 resp_bytes 73.0 orig_pkts 9 resp_pkts 6
```

Processing Step 2
Tokenized Log Sequence
[CLS] ts ···· ##04 [MASK] des ···· ##s 6 [SEP]
Token Embedding
$T_{[CLS]}$ $T_{ts}$ ···· $T_{\#\#04}$ $T_{[MASK]}$ $T_{des}$ ···· $T_{\#\#s}$ $T_6$ $T_{[SEP]}$
Segment Embedding
$S_{[CLS]}$ $S_{ts}$ ···· $S_{\#\#04}$ $S_{[MASK]}$ $S_{des}$ ···· $S_{\#\#s}$ $S_6$ $S_{[SEP]}$
Position Embedding
$P_{[CLS]}$ $P_{ts}$ ···· $P_{\#\#04}$ $P_{[MASK]}$ $P_{des}$ ···· $P_{\#\#s}$ $P_6$ $P_{[SEP]}$
Representation
$R_{[CLS]}$ $R_{ts}$ ···· $R_{\#\#04}$ $R_{[MASK]}$ $R_{des}$ ···· $R_{\#\#s}$ $R_6$ $R_{[SEP]}$

Fig. 6. Example of BERT Input Sequence Tokenization Process

Tokenizer, as shown in Figure 6.

In processing step 3, we applied dynamic padding and static padding to determine the optimal model according to padding[19-20]. Dynamic padding optimizes the efficiency of computational resources by performing variable length padding based on the maximum sequence length within each batch. Static padding fixes all sequences to 512 tokens for stability during model training and inference.

The final BERT input sequence is generated by these three processing steps.

### 3.3 Pre-training Phase

The pipeline proposed in this paper utilizes BERT, an LLM-based pre-training model, as the base model to effectively learn the features of network log data.

BERT is based on the Transformer[21] encoder structure, which can learn contextual information bi-directionally through a self-attention mechanism. This makes it suitable for understanding complex network log patterns. We used different sizes of BERT model structures (base, medium, small, mini, and tiny) to verify the ability to learn network logs according to the model size.

In the pre-training phase, the basic patterns of normal network logs are learned through masked language model (MLM) learning. As a masking technique, we refer to the dynamic masking technique of RoBERTa[22], which uses a strategy of masking 15% of all tokens.

### 3.4 Fine-tuning Phase

In the fine-tuning phase, we designed a simple binary classifier[23] model and connected it to the end of the pre-trained model, and then fine-tuned the parameters of the entire model with labeled data. The classification head for fine-tuning consists of multiple linear layers and layer normalization. In addition, ReLU activation function and dropout of 0.3 are applied to prevent overfitting.

In this phase, we applied cross-entropy loss and gradient clipping (max_norm=1.0) to ensure learning stability. And we also applied early stopping and learning rate scheduling to improve learning efficiency.

### 3.5 Inference Phase

In the inference phase, we used test network log data to verify the anomaly detection performance of the fine-tuned model.  The test network log data are converted into an input sequence for inference through the processing steps in Section 3.2.

To quantitatively evaluate the anomaly detection performance of the model, we used accuracy, F1-score, and AUC-ROC (Area Under Curve - Receiver Operating Characteristic)as the main evaluation metrics[24,25]. In particular, we focused on

F1-score and AUC-ROC, considering the data imbalance problem.

## Ⅳ. Experiment & Result

### 4.1 Experimental HW & SW Configuration

In this section, we set up an experimental environment and conducted a performance test to verify the operation and performance of the proposed pipeline structure.

The hardware configuration is as follows.

- Intel Xeon w5-3423 CPU (12 cores, 24 threads, base 2.11 GHz, max 2.97 GHz)
- 128 GB DDR5-4400 MHz RAM
- 2x NVIDIA GeForce RTX 4090 GPUs (24GB GDDR6X each)

The OS environment is Ubuntu 22.04 LTS with WSL2, and the main software and libraries are as follows.

- Base Software Environment
  - CUDA 11.8.89
  - Python 3.8.20
  - PyTorch 2.4.0 (supports CUDA 11.8)

- Key libraries used
  - Hugging Face Transformers 4.43.2
  - torchvision 0.19.0
  - scikit-learn 1.3.2
  - tokenizers 0.19.1
  - sentencepiece 0.2.0

Based on the above environment, we conducted network log anomaly detection experiments using the BERT model.

### 4.2 Experimental Data Configuration

For this experiment, we utilized the UWF-ZeekData22 dataset based on the MITRE ATT&CK Matrix.

In the pre-training phase, we only used 9.2 million normal network log data from the entire dataset as experimental data. In contrast, in the fine-tuning phase, we need to learn both normal and abnormal log data to detect anomalies.

However, as shown in Figure 4, there is a significant data imbalance between the amounts of each tactical data. This means that simple random sampling from anomalous network log data can result in certain tactics being oversampled. To avoid these issues and provide an even sample for each tactic, we implemented two ways to organize our experimental data.

#### 4.2.1 Log-scale based Weighted Sampling

Log-scale based weighted sampling means applying a logarithmic function to the frequency of occurrence of data to calculate a weight, and then sampling data based on that weight.

We calculated the weights as in Equation (1) to avoid biased sampling for specific tactics.

$$w(c) = \log(1 + n(c)) \tag{1}$$

Where,

- w(c): Weight of a specific tactic (class) c
- n(c): Frequency of occurrence of a specific tactic (class) c
- log(): Natural logarithm function
- c: Tactic type (e.g., Discovery, Credential Access, etc.)

Next, the number of target samples for each tactic is calculated as shown in Equation (2).

$$t(c) = (w(c) / \Sigma w) \times N \tag{2}$$

Where,

- t(c): number of target samples for a specific tactic c
- w(c): weight of a specific tactic (class) c, calculated from equation (1)
- ∑w: sum of weights of all tactics
- N: sum of the target sample counts of all tactics
- c: Tactic type (e.g., Discovery, Credential Access, etc.)

For example, suppose there were 1000 Discovery tactics and 10 Credential Access tactics. To ensure a balanced selection of each tactic, we can apply log-scale weighted sampling, and then normalize by the sum of the weights. This method can significantly reduce data imbalance, reducing the difference in frequency of occurrence from 100:1 to about 2.88:1. By applying the above sampling techniques, we extracted relatively small number of samples for high-frequency tactics and a larger number of samples for low-frequency tactics.

#### 4.2.2 Strategies for Maintaining Low-frequency tactics

We defined ‘low-frequency tactics’ as those that occurred less than twice in the dataset. For example, this includes tactics like ’Persistence’ and ’Initial Access’, which are less common in real-world networks but still important. These low-frequency tactics are unlikely to be included in the training data when performing simple random sampling. So we used the following three-step data partitioning strategy

- Set aside low-frequency tactic data that occurs twice or less.
- Construct the training dataset by evenly sampling tactic data with more than two occurrences, taking into account the proportion of each tactic class.
- Add the separately stored low-frequency tactics data to the training dataset

This approach can provide the following benefits.

- Learning the underlying patterns of infrequently occurring tactics
- Providing a basis for recognizing new tactical patterns
- Increassing the ability to detect new types of tactics

### 4.3 Model Training Configuration

We used BERT models of different sizes with the following hyperparameters in experiments.

- bert-base-uncased (L=12, H=768, A=12)
- bert-medium-uncased (L=8, H=512, A=8)
- bert-small-uncased (L=4, H=512, A=8)
- bert-mini-uncased (L=4, H=256, A=4)
- bert-tiny-uncased (L=2, H=128, A=2)

Where L is the number of transformer layers, H is the hidden size, and A is the number of attention heads.

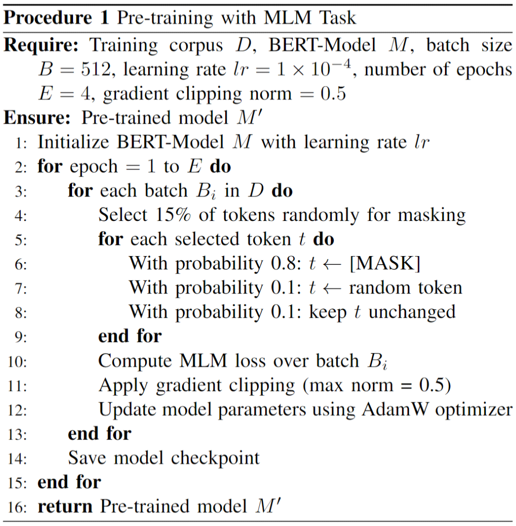
#### 4.3.1 MLM-based Pre-training Procedure

In the pre-training phase, we leveraged MLM task of BERT to learn the contextual meaning of network log patterns.

We set the pre-training hyperparameters as shown in Procedure 1. We set the batch size of the model to 512, the learning rate to $1 \times 10^{-4}$, and trained with 4 epochs.

- How to perform MLM task
  - Randomly select 15% of all tokens and mask them
  - Replace 80% of the selected tokens with [MASK] tokens
  - Replace 10% with random tokens
  - Keep the remaining 10% as original tokens

- Optimization settings for learning stability
  - Apply AdamW optimizer
  - Apply Gradient Clipping (max norm = 0.5)
  - Perform optimization by calculating the MLM loss in each batch



```
Procedure 1 Pre-training with MLM Task
Require: Training corpus $D$, BERT-Model $M$, batch size
    $B = 512$, learning rate $lr = 1 \times 10^{-4}$, number of epochs
    $E = 4$, gradient clipping norm $= 0.5$
Ensure: Pre-trained model $M'$
Initialize BERT-Model $M$ with learning rate $lr$
for epoch $= 1$ to $E$ do
    for each batch $B_i$ in $D$ do
        Select 15% of tokens randomly for masking
        for each selected token $t$ do
            With probability 0.8: $t \leftarrow$ [MASK]
            With probability 0.1: $t \leftarrow$ random token
            With probability 0.1: keep $t$ unchanged
        end for
        Compute MLM loss over batch $B_i$
        Apply gradient clipping (max norm = 0.5)
        Update model parameters using AdamW optimizer
    end for
    Save model checkpoint
end for
return Pre-trained model $M'$
```

Procedure 1. Pre-training Process

Based on the above pre-training, the learned patterns of normal network logs are used to create a pre-trained model for anomaly detection in the next step.

### 4.3.2 Fine-tuning Procedure for Binary Classification

In the fine-tuning phase, we implemented a simple binary classifier to detect network anomalies using a pre-trained BERT model for normal network log patterns only. We performed the fine-tuning process as described in Procedure 2. The hyperparameters of the fine-tuning phase were configured as follows.

In this paper, we applied various techniques to prevent overfitting and enhance the model's generalization capability. For overfitting prevention, we applied dropout (0.3) at the end of each layer and utilized gradient clipping (maximum norm = 1.0) to limit extreme weight updates. To improve the model's generalization capability, we applied layer normalization (LayerNorm) for training stabilization and applied an early stopping mechanism that monitors validation performance to prevent excessive training. Additionally, we used a reduced learning rate ($2 \times 10^{-5}$) compared to pre-training, allowing the model to achieve optimal performance through more precise weight adjustments.

```
Procedure 2 Fine-tuning for TTP-based Binary Classification
Require: Pre-trained model $M'$, labeled dataset $D'$, batch size
    $B = 512$, learning rate $lr = 2 \times 10^{-5}$, maximum epochs
    $E$, gradient clipping norm $= 1.0$
Ensure: Fine-tuned binary classifier $C$
Initialize:
    classifier_head ← [Linear(768 → 512), LayerNorm,
    ReLU, Dropout(0.3)]
                    [Linear(512 → 256), LayerNorm, ReLU,
    Dropout(0.3)]
                    [Linear(256 → 2)]
Prepare binary labels: label = 1 if tactic/technique nonzero,
    else 0
for epoch $= 1$ to $E$ do
    for each batch $B_j$ in $D'$ do
        Compute BERT representations
        Forward-pass through classification head
        Compute cross-entropy loss
        Apply gradient clipping (max norm = 1.0)
        Update parameters using AdamW optimizer
    end for
    Evaluate validation performance
    Apply early stopping if needed
end for
return Fine-tuned classifier $C$
```

Procedure 2. Fine-tuning Process

- Classifier head structure
  - 768 → 512 → 256 → 2-dimensional output
  - LayerNorm, ReLU activation function, and dropout (0.3) are applied at the end of each layer
- Training settings
  - Batch size: 512 (same as pre-training)
  - Learning rate: $2 \times 10^{-5}$ (reduced from pre-training)
  - Applying gradient clipping (max norm = 1.0)
  - Using AdamW optimizer
  - Applying a loss function based on cross-entropy loss
  - Computing BERT representation on each batch and performing forward-path through the classification head
- Optimizing model performance
  - Applying early stopping by monitoring validation performance
- Labeling
  - True (1) if tactic, or False (0) if not.

Through this fine-tuning process, we validated the anomaly detection performance of the network log pre-trained model.

## 4.4 Performance Evaluation Results

In this paper, we evaluated the performance of the proposed pipeline implementation using accuracy, F1-score, and AUC-ROC as metrics.

We analyzed the performance of MLM-based pre-training task in section 4.4.1. Then, we evaluated the performance of the binary classification task for network anomaly detection in section 4.4.2. In both experiments, we explored the impact of dynamic and static padding on anomaly detection performance and also analyzed the optimal model according to model size.

### 4.4.1 Pre-training Phase Performance

In the pre-training phase, the MLM task was applied to the BERT model to effectively learn the complex spatio-temporal and connectivity patterns of network logs, and the results are shown in Table 3.

Table 3. Pre-training Performance Evaluation

| Model | Padding | Pre-Training | | |
|---|---|---|---|---|
| | | Loss | Mask_Pred | Mask_Error |
| BERT-Base | Dynamic | 0.3200 | 0.9185 | 0.0785 |
| | Static | 0.3354 | 0.9180 | 0.0788 |
| BERT-Medium | Dynamic | 0.2908 | 0.9228 | 0.0726 |
| | Static | 0.3070 | 0.9220 | 0.0734 |
| BERT-Small | Dynamic | 0.3139 | 0.9175 | 0.0794 |
| | Static | 0.3277 | 0.9169 | 0.0799 |
| BERT-Mini | Dynamic | 0.3194 | 0.9162 | 0.0807 |
| | Static | 0.3243 | 0.9158 | 0.0810 |
| BERT-Tiny | Dynamic | 0.3302 | 0.9087 | 0.0869 |
| | Static | 0.3326 | 0.9082 | 0.0873 |

Table 3 shows that dynamic padding outperforms static padding for all sizes of models used in the experiment. This suggests that dynamic padding can handle variable-length network log sequences more effectively. Furthermore, all models achieved a mask prediction accuracy of 0.9 or higher, regardless of the size or structure of the model. These results demonstrate that the MLM tasks can effectively learn patterns from network log sequences.

#### 4.4.2 Fine-tuning and Inference Phase Performance

We applied log-scale-based weighted sampling and low-frequency tactic preservation strategy to deal with data imbalance issues in the fine-tuning and inference phases. We then performed a binary classification task to detect network anomalies. Table 4 shows the performance evaluation results of the fine-tuned model.

As shown in Table 4, the fine-tuned model achieved a high accuracy of over 0.94, an AUC-ROC value close to 1.0, and an F1-score of over 0.93 for all configurations. This demonstrates that the pre-trained model can be effectively applied to the specific task of network anomaly detection.

In addition, unlike the pre-training phase, we found that the optimal padding varied by model size during the fine-tuning and inference phases. Static padding performed better for large models (BERT-Base, Medium), while dynamic padding performed better for small models (BERT-Small, Mini, Tiny). This suggests that the choice of model size and padding method can be an important consideration when implementing network anomaly detection systems.

Table 4. Inference Performance Evaluation

| Model | Padding | Accuracy | F1-Score | AUC-ROC |
|---|---|---|---|---|
| BERT-Base | Dynamic | 0.9860 | 0.9822 | 1.0000 |
| | Static | 0.9867 | 0.9830 | 1.0000 |
| BERT-Medium | Dynamic | 0.9513 | 0.9351 | 1.0000 |
| | Static | 0.9800 | 0.9743 | 1.0000 |
| BERT-Small | Dynamic | 0.9853 | 0.9813 | 1.0000 |
| | Static | 0.9760 | 0.9691 | 0.9999 |
| BERT-Mini | Dynamic | 0.9867 | 0.9830 | 1.0000 |
| | Static | 0.9586 | 0.9454 | 1.0000 |
| BERT-Tiny | Dynamic | 0.9927 | 0.9907 | 1.0000 |
| | Static | 0.9480 | 0.9304 | 0.9999 |

## Ⅴ. Conclusion

This paper proposed a pipeline based on a pre-training model for detecting anomalies in network logs. To verify the applicability of the proposed pipeline, we utilized the UWF-ZeekData22 dataset generated based on the MITRE ATT&CK Matrix.

To generate the dataset for training, we first extracted key features from network logs through EDA, then grouped the extracted features for feature engineering and preprocessing, and finally constructed the input sequences for pre-training the BERT model.

In the pre-training phase, we verified that the model effectively learns the complex patterns of network logs. In the fine-tuning and inference phase, we showed that the data imbalance problem can be solved by log-scale-based weighted sampling and low-frequency tactic preservation strategies, which eventually allows the model to achieve balanced high performance for different tactics.

The evaluation results showed that all models performed well, with accuracies above 0.94 and AUC-ROC approaching 1.0. We also found that the optimal padding tended to vary depending on the model size. Static padding performed better on large models and dynamic padding performed better on small models. These results suggest that the choice of model size and padding method can be important considerations when constructing network anomaly detection models.

While we were unable to make direct performance comparisons due to a lack of research on existing detection models with similar approaches, we were encouraged by the high performance of our proposed pipeline implementation. We also found that there was a correlation between the padding method and model size, and we recommend that future studies take this into account when designing models.

From this point of view, the proposals in this paper are expected to provide practical case study in the field of modernization of network security systems and development of real-time security monitoring solutions.

This paper is an initial study for the basic baseline of a LLM(BERT) pre-training pipeline using network traffic data based on the MITRE ATT&CK framework. This study focused on validating the core features and performance of the anomaly detection model and verifying that the proposed system works as intended.

In future work, we plan to conduct a comprehensive evaluation of practical aspects such as deployability in real-world environments, real-time performance, and computational cost analysis. In an attack scenario involving the stages of penetration (initial access, privilege escalation, internal reconnaissance, data exfiltration, etc.) that may occur in real-world network environments, we aim to learn and collectively analyze the subtle changes in log patterns for each stage. In addition, we plan to investigate the practical applicability by conducting research on trade-offs of BERT variants and developing effective tokenization techniques to identify attacks with higher accuracy than previous single-event-based detection systems. Additionally, we are considering the possibility of developing a security detection model utilizing the Retrieval Augmented Generation (RAG) approach. By utilizing knowledge databases based on publicly available security vulnerability information such as the Common Weakness Enumeration (CWE) list, we plan to develop more advanced security threat detection models through a combined approach of sLLM and RAG. Based on the above research, we aim to go beyond simply detecting anomalous behavior to demonstrate the practicality of the proposed model and develop it into a solution that meets practical needs in the security field.

From this point of view, the proposals in this paper are expected to provide practical case study in the field of modernization of network security systems and development of real-time security monitoring solutions.

## References

[1] MITRE Corporation, "*MITRE ATT&CK*,"(2025), Retrieved Feb. 1, 2025, from https://attack.mitre.org

[2] University of West Florida, "*UWF-ZeekData22*," (2025), Retrieved Feb. 1, 2025, from https://datasets.uwf.edu/

[3] UCI KDD Archive, "*KDD Cup 1999*,"(1999), Retrieved Feb. 1, 2025, from http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[4] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. Second IEEE Symp. Comput. Intell. Secur. Def. Appl.*, pp. 1-6, Ottawa, ON, Canada, Jul. 2009. (https://doi.org/10.1109/CISDA.2009.5356528)

[5] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. MilCIS*, pp. 1-6, Canberra, Australia, Nov. 2015. (https://doi.org/10.1109/MilCIS.2015.7348942)

[6] G. Maciá-Fernández, J. Camacho, R. Magán-Carrión, P. García-Teodoro, and R. Therón, "UGR'16: A new dataset for the evaluation of cyclostationarity-based network IDSs," *Comput. Secur.*, vol. 73, pp. 411-424, Mar. 2018. (https://doi.org/10.1016/j.cose.2017.11.004)

[7] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "A detailed analysis of the CICIDS2017 data set," in *Proc. ICISSP*, pp. 172-188, Funchal, Portugal, Jan. 2018. (https://doi.org/10.1007/978-3-030-25109-3_9)

[8] X. Du and K. Tanaka-Ishii, "Correlation dimension of natural language in a statistical manifold," *Phys. Rev. Res.*, vol. 6, L022028, May 2024. (https://doi.org/10.1103/PhysRevResearch.6.L022028)

[9] H. Guo, X. Lin, J. Yang, et al., "TransLog: A unified transformer-based framework for log anomaly detection," *arXiv preprint arXiv: 2201.00016*, Jan. 2022. (https://doi.org/10.48550/arXiv.2201.00016)

[10] E. Karlsen, X. Luo, N. Zincir-Heywood, and M. Heywood, "Benchmarking large language models for log analysis, security, and interpretation," *J. Netw. Syst. Manag.*, vol. 32, no. 59, Jun. 2024. (https://doi.org/10.1007/s10922-024-09831-x)

[11] S. M. T. Far and F. Feyzi, "Large language models for software vulnerability detection: A guide for researchers on models, methods, techniques, datasets, and metrics," *Int. J. Inf. Secur.*, vol. 24, no. 78, Feb. 2025. (https://doi.org/10.1007/s10207-025-00992-7)

[12] J. Sundin and L. Särud, "AI-driven log analysis for intrusion detection," M.S. Thesis, Dept. of Automatic Control, Lund University, Sep. 2024. Retrieved Feb. 27, 2025, from https://lup.lub.lu.se/student-papers/search/publication/9173491

[13] S. Bagui, D. Mink, S. Bagui, T. Ghosh, T. McElroy, E. Paredes, N. Khasnavis, and R. Plenkers, "Detecting reconnaissance and discovery tactics from the MITRE ATT&CK framework in zeek conn logs using spark's machine learning in the big data framework," *Sensors*, vol. 22, no. 20, p. 7999, Oct. 2022. (https://doi.org/10.3390/s22207999)

[14] S. S. Bagui, D. Mink, S. C. Bagui, M. Plain, J. Hill, and M. Elam, "Using a graph engine to visualize the reconnaissance tactic of the MITRE ATT&CK framework from UWF-ZeekData22," *Future Internet*, vol. 15, no. 7, 236, Jul. 2023. (https://doi.org/10.3390/fi15070236)

[15] S. S. Bagui, D. Mink, S. C. Bagui, D. H. Sung, and F. Mahmud, "Graphical representation of UWF-ZeekData22 using memgraph," *Electronics*, vol. 13, no. 6, p. 1015, Mar. 2024. (https://doi.org/10.3390/electronics13061015)

[16] S. S. Bagui, D. Mink, S. C. Bagui, and S. Subramaniam, "Resampling to classify rare attack tactics in UWF-ZeekData22," *Knowledge*, vol. 4, no. 1, pp. 96-119, Jan. 2024. (https://doi.org/10.3390/knowledge4010006)

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. 2019 Conf. North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 4171-4186, Minneapolis, MN, USA, Jun. 2019. (https://doi.org/10.18653/v1/N19-1423)

[18] K. S. Nugroho, A. Y. Sukmadewa, and N. Yudistira, "Large-scale news classification using BERT language model: Spark NLP approach," in *Proc. 6th Int. Conf. SIET 2021*, pp. 240-246, Yogyakarta, Indonesia, Sep. 2021. (https://doi.org/10.1145/3479645.3479658)

[19] A. L. Rio, M. J. Martin, A. Perera-Lluna, and R. Saidi, "Effect of sequence padding on the performance of deep learning models in archaeal protein functional prediction," *Sci. Rep.*, vol. 10, no. 14634, Sep. 2020. (https://doi.org/10.1038/s41598-020-71450-8)

[20] A. Kundu, R. D. Lee, L. Wynter, R. K. Ganti, and M. Mishra, "Enhancing training efficiency using packing with flash attention," *arXiv preprint arXiv:2407.09105*, Sep. 2024. (https://doi.org/10.48550/arXiv.2407.09105)

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. 31st Int. Conf. NIPS'17*, pp. 6000-6010, Long Beach, CA, USA, Dec. 2017. (https://dl.acm.org/doi/10.5555/3295222.3295349)

[22] Y. Liu, et al., "RoBERTa: A robustly

optimized BERT pretraining approach," *arXiv preprint arXiv:1907.11692*, Jul. 2019. (https://doi.org/10.48550/arXiv.1907.11692)

[23] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune BERT for text classification?," in *Proc. CCL 2019*, LNCS, vol. 11856, pp. 194-206, Kunming, China, Oct. 2019. (https://doi.org/10.1007/978-3-030-32381-3_16)

[24] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. & Manag.*, vol. 45, no. 4, pp. 427-437, Jul. 2009. (https://doi.org/10.1016/j.ipm.2009.03.002)

[25] J. Davis and M. Goadrich, "The relationship between precision-recall and ROC curves," in *Proc. 23rd ICML '06*, pp. 233-240, Pittsburgh, USA, Jun. 2006. (https://doi.org/10.1145/1143844.1143874)

### Kyuchang Kang

Feb. 1997 : M.S. in Dept. of Electronics Engineering, Kyungpook National University

Aug. 2009 : Ph.D. in Dept. of Computer Engineering, Chungnam National University

Feb. 1997~Mar. 2001 : Researcher, Agency for Defense Development (ADD)

Mar. 2001~Mar. 2017 : Principal Researcher, SW· Content Laboratory, Electronics and Telecommunications Research Institute (ETRI)

Mar. 2017~Current : Associate Professor, Dept. of IT and Communication Convergence Engineering, Kunsan National University

<Research Interest> Open Software Platform, Artificial Intellgence of Things(AIoT), Artificial Intellgence security (AI security), Human-Understanding Cognitive Computing

[ORCID:0000-0003-0833-8906]

### Yu-Jin So

Feb. 2023 : B.S. in IT Convergence & Communication Eng. Major, School of IT Information & Control Eng., Kunsan National University

Feb. 2025 : M.S in Information Communication Radio Engineering Major, School of Electronics and Information Eng.

June. 2025~Current : Researcher, Intelligent Network Security Research Section, Electronics and Telecommunications Research Institute (ETRI)

<Research Interest> Artificial Intelligence, Artificial Intellgence security (AI security)

[ORCID:0009-0005-9413-0715]

### Jong-Geun Park

Feb. 1999 : M.S. in Dept. of Industrial Engineering, Sungkyunkwan University

Feb. 2013 : Ph.D. in Dept. of Computer Engineering, Chungnam National University

Mar. 1999~Apr. 2001 : Researcher, Agency for Defense Development (ADD)

May. 2001~Current : Director/Principal Researcher, Intelligent Network Security Research Section, Electronics and Telecommunications Research Institute (ETRI)

<Research Interest> Mobile Communication Security, Cloud Security, Defense Cybersecurity, AI Security etc.

[ORCID:0000-0003-4973-7700]