The Journal of Korean Institute of Communications and Information Sciences '25-04 Vol.50 No.04
https://doi.org/10.7840/kics.2025.50.4.647

A Study on Data Reconstruction and Model Parameter Optimization for Implementation of Anomaly Detection System Based on User Behavior Analysis

Yu-Jin So[♦], Jong-Geun Park[*], Kyuchang Kang[°]

ABSTRACT

In this paper, we implemented an anomaly detection system based on user behavior analysis to effectively detect user anomalies in a specific domain. For this purpose, we performed EDA on User Behavior Data, defined feature factors for user behavior features in feature engineering, and proposed a method for combining feature factors. We performed preprocessing and vectorization on the session data to define user behavior patterns as 'Session' and provide them as input to the model. The vectorized session data was pre-trained on BERT Model Architecture using only normal session data. We performed an anomaly detection performance evaluation after fine-tuning using normal and abnormal session data. As a result of the performance evaluation, BERT-Medium-uncased model performed well with an Accuracy of 0.9630 and an F1-Score of 0.9628, and the overall performance was balanced. As a result, we confirmed that by utilizing EDA and feature engineering for data, we can effectively perform pre-training and fine-tuning and implement a high-performance anomaly detection system.

Key Words : User behavior analysis, Feature engineering, Feature factor combination method, User anomaly detection system

Ⅰ. Introduction

In recent years, advanced technologies at the core of the Fourth Industrial Revolution, such as big data, artificial intelligence (AI), the Internet of Things (IoT), cloud computing, and 5G network communications, have been widely introduced and made available. These advanced technologies are closely connected to our society and have a profound impact on our daily lives and various industries. However, as these advanced technologies evolve, potential cybersecurity threats also increase.

The issue of cybersecurity threats is becoming more prominent not only domestically but also internationally. In the 2024 Verizon Data Breach Investigations Report (DBIR)[1], researchers found that 68% of data breaches are linked to non-malicious human factors. Additionally, Proofpoint's "2024 Voice of the CISO[2]" report, which surveyed 1,600 chief information security officers (CISOs) worldwide, found that 70% of CISOs believe their organizations are at risk of suffering a significant cybersecurity attack within the next year. Among the various cybersecurity threats, CISOs expect human error to

※ 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 Author : Kunsan National University, School of Electronics and Information Eng., so.npng98@gmail.com, 학생회원
° Corresponding Author : Kunsan National University, IT Convergence & Communication Eng., kc.kang@kunsan.ac.kr, 정회원
* Electronics and Telecommunications Research Institute (ETRI), queue@etri.re.kr, 정회원
논문번호 : 202410-220-A-RN, Received September 30, 2024; Revised December 3, 2024; Accepted December 7, 2024

be the most significant cybersecurity vulnerability, as insider threats and human-caused data loss continue to rise. In 2024, 74% of CISOs agreed that human error is a serious cybersecurity threat, up 18% from two years ago, compared to 60% in 2023 and 56% in 2022. Additionally, 80% of CISOs recognize cybersecurity threats caused by human error, such as data loss due to employee carelessness, as a significant issue requiring attention over the next two years. This is a 17% increase from the 63% reported in 2023, suggesting that insider threats can occur independent of malicious intent and individual intent. This means that organizations must also have countermeasures in place against insider threats alongside building technical defenses against cybersecurity threats. This emphasizes that insider threats are just as important a security risk as external attacks.

Businesses and public institutions are working to strengthen their cyber defenses to respond to and prevent cybersecurity threat incidents. These efforts include protecting sensitive information through data encryption and access control, improving internal security systems by introducing solutions from external specialized companies, and establishing continuous monitoring systems for real-time threat detection. As the importance of cybersecurity is increasing due to the diversification of attack types such as ransomware, insider threats, and DDoS attacks, each organization needs to establish a systematic and effective advanced security system to respond to cybersecurity threats.

In general, traditional security software, such as firewalls and intrusion detection systems, can effectively detect and respond to well-known cyberattack patterns. However, these systems have limitations in identifying and preventing new types of advanced cybersecurity threats that are constantly evolving. In particular, the increasing number of data security incidents caused by human error, such as data leaks, mistakes, and negligence by insiders, shows that cybersecurity threats are not only coming from external factors but also from internal factors. Currently, various organizations are making great efforts to utilize various security systems and strengthen cybersecurity. However, in order to effectively respond to newly modified attack types or potential internal cybersecurity threats in addition to existing generalized cyber threats, existing security systems alone are not enough. Along with existing security systems, a new cybersecurity threat detection system is needed that can respond to new types of attacks or potential internal threats.

In this paper, we aim to implement a user behavior analysis-based anomaly detection system for detecting abnormal user behavior in a specific domain. For this purpose, we performed exploratory data analysis (EDA) on a user behavior dataset to extract user behavior features and define user behavior feature elements. In the process, we proposed a new token combination method for anomaly detection and defined user behavior patterns as “sessions”. Next, we performed input sequence vectorization on the session data after a simple preprocessing. We then leveraged BERT model architecture as the underlying model for our user behavior analysis-based anomaly detection system. During the pretraining process, we used only normal session data for the vectorized session data. In this study, we focused on the implementation of the anomaly detection system pipeline as our primary goal. For this reason, we added a simple binary classifier during the fine-tuning and system evaluation phases to perform performance evaluation. We aim to explore a practical methodology for implementing an anomaly detection system, explicitly emphasizing analyzing user behavior data characteristics and investigating optimal model architectures for this domain. To achieve this goal, we propose a comprehensive approach addressing two main aspects of system implementation. First, we conduct an in-depth analysis of user behavior characteristics to identify essential features to consider in anomaly detection. We propose a novel feature factor combination method as part of our feature engineering approach to address the challenge of processing large-scale chunk data into BERT-compatible input. This method enables effective tokenization of user behavior information while preserving meaningful patterns within the data. Furthermore, we conduct extensive ablation case studies on BERT model to determine the most efficient configuration for anomaly detection. This includes exploring various hyperparameters for pre-training and transfer learning phases to optimize the learning process for our specific input data characteristics. We systematically investigate different BERT model architectures to identify the optimal model size that balances detection performance with computational efficiency.

This paper is organized as follows. Chapter 2 reviews existing related work, and Chapter 3 details the structure of our proposed user behavior analysis-based anomaly detection system. Chapter 4 describes the experimental results of the implemented user anomaly detection system, and finally, Chapter 5 discusses conclusions and future work.

Ⅱ. Related Works

Anomaly detection techniques[3,4] refer to defining patterns considered “normal” and identifying outliers or abnormal patterns in data that deviate from them. Anomaly detection techniques can distinguish normal samples from abnormal (outliers, anomalies) samples to find data within a data set with different characteristics from other observations. Anomaly detection techniques must now go beyond applying predefined rules. It can automatically learn new types of anomalies, such as real-time anomaly detection in data streams and pattern recognition in complex multivariate data. As technology evolves, the research and application of anomaly detection techniques are expanding from statistical methods to machine learning[5,6] and deep learning[7,8] approaches.

One-class support vector machine (OCSVM)[9] is an unsupervised learning model variant of supervised learning SVM. In anomaly detection, OCSVM[10-13] trains by mapping normal data into a high-dimensional space and finding the normal data’s minimum boundary. It detects an anomaly if the new data is not contained within the boundary.

A convolutional neural network (CNN)[14] is a deep learning model that uses convolutional layers to extract and train image features. In anomaly detection, CNN[15-17]learns features from normal data, and if features extracted from new input data are different from the trained data, it detects an anomaly.

Long-Short-Term Memory (LSTM)[18] is a recurrent neural network (RNN) model that specializes in processing time series data. In anomaly detection, LSTM[19-22] learns normal time series data patterns. Based on the trained patterns, LSTM predicts the value at the next point in time and calculates the error between the predicted value and the actual observed value. It detects the data as an anomaly if the error exceeds a threshold.

Auto-Encoder (AE)[23] is a neural network-based unsupervised learning model with an encoder-decoder structure. It is mainly used for data compression and feature extraction. Encoders compress the input data into a low-dimensional representation. The low-dimensional representation is placed in the AE’s middle layer (latent space). The decoder reconstructs the latent space’s low-dimensional representation into the original dimensions. The AE trains the main features of the data in this process. The AE[24-26] trains a model on normal data in anomaly detection. The trained AE calculates the errors that occur in reconstructing the input data. In this case, if the reconstruction error of the input data is significant, it is detected as an anomaly.

A Variational Auto-Encoder (VAE)[27] is an unsupervised learning model with an encoder-decoder structure that learns the probability distribution of data. Encoders compress the input data into a probability distribution defined by a mean and variance. Decoders recover the samples’ original dimension from the latent space’s probability distribution. VAE represents the latent variables as probability distributions and uses reconstruction error and KL(Kullback–Leibler) divergence as loss functions. In anomaly detection, a VAE[28-31] trained on normal data calculates the reconstruction error on the input data. If the reconstruction error of the input data is significant, it detects an anomaly.

In recent years, transformer-based models[32] such as BERT have demonstrated outstanding performance in natural language processing, and attempts have been made to utilize them in anomaly detection. BERT’s[33] bidirectional encoding structure and self-attention mechanism can effectively capture long-term dependencies and complex patterns in sequence data and have shown remarkable results in anomaly detection tasks.

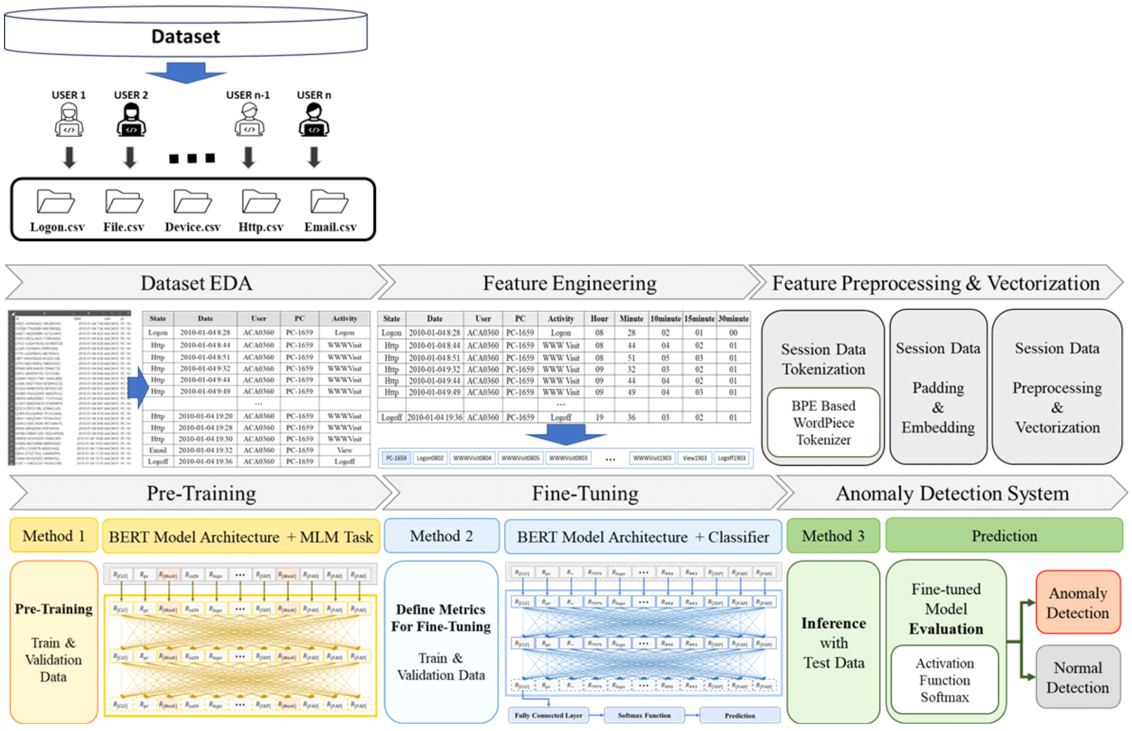
Dang et al. (2021)[34] proposed a BERT-based model, TS-BERT, for anomaly detection in time series data. TS-BERT was developed to effectively handle the long-term dependence of time series data and improve the problem of lack of label data. However, due to BERT structure, TS-BERT suffered from increased computational complexity in mapping low-dimensional data to high-dimensional space and long training time. In addition, label generation using the spectral residual method made fully unsupervised learning impossible, and it limited itself to fully reflecting the unique features of time series data.

Guo et al. (2021)[35] proposed a BERT-based model, LogBERT, for anomaly detection in log data. LogBERT was designed to detect log data anomalies by performing pre-training with Masked Log Key Prediction (MLKP) and fine-tuning for anomaly detection. However, LogBERT suffered from high computational complexity and increased processing time due to BERT structure, performance deviations depending on the features of the pre-training data, and limitations in interpreting the results of deep learning models.

Tang and Guan (2024)[36] proposed a BERT-based model, SD-BERT, to detect anomalies in system log data. SD-BERT was designed to capture log sequences' global context and local features effectively by introducing a Separated Score Attention (SSA) mechanism and a dual branching module. However, SD-BERT was trained only on normal log sequences, which made it difficult to detect new types of anomalies, model complexity limited real-time processing, and SSA and the dual branching module were optimized for specific datasets, which limited generalization.

Ⅲ. Method

The structure of the user anomaly detection system based on user behavior analysis is shown in Figure 1, which consists of (1) exploratory analysis of user behavior dataset (Dataset EDA), (2) behavior feature extraction (Feature Extraction/Engineering), (3) fea-


Dataset
USER 1
USER 2
USER n-1
USER n
Logon.csv
File.csv
Device.csv
Http.csv
Email.csv
Dataset EDA
Feature Engineering
Feature Preprocessing & Vectorization
Session Data Tokenization
BPE Based WordPiece Tokenizer
Session Data Padding & Embedding
Session Data Preprocessing & Vectorization
Pre-Training
Fine-Tuning
Anomaly Detection System
Method 1
BERT Model Architecture + MLM Task
Method 2
BERT Model Architecture + Classifier
Method 3
Prediction
Pre-Training Train & Validation Data
Define Metrics For Fine-Tuning
Train & Validation Data
Fully Connected Layer
Softmax Function
Prediction
Inference with Test Data
Fine-tuned Model Evaluation
Activation Function Softmax
Anomaly Detection
Normal Detection

State	Date	User	PC	Activity
Logon	2010-01-04 8:28	ACA0360	PC-1659	Logon
Http	2010-01-04 8:44	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 8:51	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 9:32	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 9:44	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 9:49	ACA0360	PC-1659	WWWVisit
		…		
Http	2010-01-04 19:20	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 19:28	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 19:30	ACA0360	PC-1659	WWWVisit
Email	2010-01-04 19:32	ACA0360	PC-1659	View
Logoff	2010-01-04 19:36	ACA0360	PC-1659	Logoff

State	Date	User	PC	Activity	Hour	Minute	10minute	15minute	30minute
Logon	2010-01-04 8:28	ACA0360	PC-1659	Logon	08	28	02	01	00
Http	2010-01-04 8:44	ACA0360	PC-1659	WWW Visit	08	44	04	02	01
Http	2010-01-04 8:51	ACA0360	PC-1659	WWW Visit	08	51	05	03	01
Http	2010-01-04 9:32	ACA0360	PC-1659	WWW Visit	09	32	03	02	01
Http	2010-01-04 9:44	ACA0360	PC-1659	WWW Visit	09	44	04	02	01
Http	2010-01-04 9:49	ACA0360	PC-1659	WWW Visit	09	49	04	03	01
				…					
Logoff	2010-01-04 19:36	ACA0360	PC-1659	Logoff	19	36	03	02	01

PC-1659	Logon0802	WWWVisit0804	WWWVisit0805	WWWVisit0903	…	WWWVisit1903	View1903	Logoff1903

Fig. 1. Anomaly detection system based on user behavior analysis architecture

ture preprocessing and vectorization, (4) pre-training and fine-tuning, and (5) anomaly detection system evaluation.

First, Dataset EDA performs a preliminary analysis of user activity log files. Next, Feature Extraction/Engineering extracts features describing user behavior and generates feature vectors through feature factor combination. Then, Feature Preprocessing & Vectorization create sessions with preprocessed and vectorized user behavior patterns. The created sessions are used as input for BERT model. Next, pre-training is performed only during normal sessions, and fine-tuning is performed using anomaly detection tasks. Finally, the anomaly detection system with pre-trained and fine-tuned weights is evaluated using separate test data.

3.1 Dataset

Computer Emergency Response Team (CERT)[37] dataset is an insider threat behavior dataset provided by the Software Engineering Institute (SEI) at Carnegie Mellon University in the United States. The CERT dataset contains user activity logs and profiles spanning 18 months. The CERT dataset is widely used in the cybersecurity and cyber threat fields for analyzing user behavior patterns and evaluating the performance of anomaly detection algorithms. The anomaly detection system implementation in this paper uses the CERT r6.2 dataset.

The CERT r6.2 version is a large dataset of about 93 GB. This dataset includes 4,000 users, of which only 5 are malicious insiders, a tiny percentage. There are a total of 5 malicious insider scenarios, with one malicious scenario for each malicious insider. The r6.2 version of CERT includes eight files, including a Lightweight Directory Access Protocol (LDAP) file containing user information and five activity log (logon.csv, file.csv, device.csv, Http.csv, and email.csv) files related to user behavior. This dataset is highly imbalanced, with 135,117,169 total activity logs for all users but only 470 malicious activity logs from insiders.

Table 1 describes the overall basic statistics of the CERT r6.2 dataset, and Table 2 describes the malicious insider scenarios in the dataset. Table 3 describes the files in the dataset, and Table 4 summarizes the key fields in the activity log files in the dataset.

Table 1. The statistics of CERT r6.2 dataset

Dataset	r6.2
Employees	4000
Role	46
Insiders	5
Activity Log File	5
Activity	135,117,169
Anomaly Activity	470

Table 2. Malicious insider scenarios in the CERT r6.2 dataset

Scenario	Description
Scenario 1 ACM2278	A user who has not previously used removable drives uses a removable drive after hours, moves their data elsewhere, and retires.
Scenario 2 CMP2946	An employee who browses job search websites and applies for jobs at competitors, and steals data from removable drives with increasing frequency.
Scenario 3 PLJ1771	A disgruntled system administrator steals a keylogger from a supervisor's computer, sneaks in as the supervisor the next day, sends a mass email to the company, and leaves.
Scenario 4 CDE1846	An employee logs into another employee's computer and sneaks in his own email when he finds something of interest.
Scenario 5 MBG3183	A user who was laid off uploads documents to Dropbox for personal gain uploading documents to Dropbox for personal gain.

Table 3. CERT r6.2 dataset file information

File	Description
Logon.csv	Records the user's connection status
File.csv	Records work-related files (Ex. WWWDOWNLOAD, WWWVISIT etc.)

File	Description
Device.csv	Record contents of removable device and thumb drive usage
Http.csv	Records everything related to web browsing
Email.csv	Records 5 different log activities for every employee in the virtual company
Decoy.csv	Records the list of decoy file and the PC name of the decoy file user (Ex. filename : C:\LJE2413\795JW126.jpg, PC : PC-0302)
Psychometric.csv	Records employee personality
LDAP	Records user information such as job title, department, work period, etc.

Table 4. CERT r6.2 dataset activity log files

File	Field	Activity
Logon.csv	id, date, user, pc, activity	Logon, Logoff
File.csv	id, date, user, pc filename, activity, content, to_removable_media, from_removable_media	File Open, File Delete, File Copy, File Write
Device.csv	id, date, user, pc, file_tree, activity	Connect, Disconnect
Http.csv	id, date, user, pc, url, activity	WWW Visit, WWW Upload, WWW Download
Email.csv	id, date, user, pc, to, cc, bcc, from, activity, size, attachments	Send, View

3.2 User Behavior Dataset EDA

Exploratory data analysis (EDA)[38,39] is an analysis method performed in the early stages of data analysis. The goal of EDA is to understand the structure and features of the data. In the process, relationships between variables are explored, and patterns in the data are discovered. By performing EDA, insights can be gained for future analysis direction or effective modeling, and analysis of user activity logs is required to detect user anomalies effectively. In this paper, we performed EDA on raw activity log files ('Logon.csv', 'File.csv', 'Device.csv', 'Http.csv', and 'Email.csv') for all users in the CERT r6.2 dataset to identify user behavior types and behavior patterns.

After EDA, we summarized the key features to consider in this dataset as follows;

(a) Everyone has a 'Logon.csv' file corresponding to the user's commute file, but other activity log files may or may not exist.
(b) Users' behavior patterns are represented by time series data, with periodicity in the form of log-on-logoff.
(c) Each user normally works a different day and time of day for their job.
(d) Each user is assigned a computer, but they occasionally use a public computer to access other users' computers.
(e) If a user's job is ITAdmin, he/she can freely access other users' computers as a system administrator.
(f) As a system administrator, an ITAdmin is likely to have many entries in their activity logs for accessing other users' computers.

As seen above, the absence of a specific behavior cannot be considered abnormal because of cases such as (a). For example, a user may have never opened a file, but if the lack of file-related behavior is considered an anomaly, it will be recognized as a false positive. From (c), we can see different normal behavior patterns for each occupation. We can see from (e) and (f) that the ITAdmin occupation has a unique behavior pattern, unlike other occupations. To avoid detecting different behavior patterns of various users as abnormal behavior, we need to define the behavior patterns of users, and for this, we need feature factors that determine the behavioral characteristics of users.

3.3 Behavior Feature Extraction

For a pre-trained model to properly understand and train on user behavior patterns, it is important to provide user behavior data as input to the model using meaningful feature factors. In this paper, we extract meaningful feature fields from user behavior feature fields and define feature factor, which means a factor that determines a feature. By defining meaningful feature factors, we can reduce the probability of false positives, which is mistakenly recognizing normal behavior as abnormal behavior. In addition, we can expect to improve the performance of the pre-trained model. Table 5 describes the fields extracted by EDA on the user behavior data. Table 6 shows the user behavior patterns with periodicity in the form of logon-logoff defined as a result of EDA.

Table 5. Fields extracted from user behavior data EDA

Field	Description
State	Field to distinguish activity logs
Date	YYYY-MM-DD HH:mm:SS
User	User ID
PC	Personal Computer PC Name
Activity	User Activity Logs

Table 6. Behavioral pattern from user behavior data EDA

State	Date	User	PC	Activity
Logon	2010-01-04 08:28	ACA0360	PC-1659	Logon
Http	2010-01-04 08:44	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 08:51	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 09:32	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 09:44	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 09:49	ACA0360	PC-1659	WWWVisit
		…		
Http	2010-01-04 19:20	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 19:28	ACA0360	PC-1659	WWWVisit
Http	2010-01-04 19:30	ACA0360	PC-1659	WWWVisit
Email	2010-01-04 19:32	ACA0360	PC-1659	View
Logoff	2010-01-04 19:36	ACA0360	PC-1659	Logoff

3.3.1 Associated Field Extraction

The results of extracting user behavior patterns by performing EDA above are shown in Tables 5 and 6. Five activity log files ('Logon.csv', 'File.csv', 'Device.csv', 'Email.csv', 'Http.csv') are associated with user behavior in the experimental data. However, some users only have activity records in 'Logon.csv'. That is, they do not perform any activities except login and logoff. Considering this situation, we extracted only the 'Date', 'PC', and 'Activity' fields common in the five activity log files as feature factor fields associated with user behavior data. We defined feature factors that carry unique significance in anomaly behavior detection based on user information. Specifically, we selected three core feature factors from the CERT dataset: time, location, and behavior information, as described in Table 7. 'Date' field represents the 'Time' feature factor indicating when an action occurred, 'PC' field represents the 'Location' feature factor indicating where an action took place, and 'Activity' field represents the 'Behavior' feature factor indicating what action occurred.

Table 7. Fields associated with user behavior feature

Field	Definition	Description
Date	Time Feature Factor	Indicates when an action occurred
PC	Location Feature Factor	Indicates where an action took place
Activity	Behavior Feature Factor	Indicates what action occurred

3.3.2 Feature Factor Combination

In this chapter, we propose a feature factor combination method to detect user anomalies effectively. A token with a single piece of information may not sufficiently represent a user's complex behavior patterns. For example, a simple 'Behavior' feature factor does not capture the time or location context of the behavior. Also, 'Time' feature factor alone does not reveal which behavior occurred at a specific time of day. In this paper, we use a combination of feature factors to generate meaningful tokens optimized for anomaly detection, while exploring effective pre-processing methods to provide optimal input tokens to the model. Figure 2 shows how the input data of the model is created by the feature factor combination method.

In Step 1, we explain the process of extracting elements from the 'Date' field to be used as Time feature factors. First, the user behavior patterns for the 'Logon-Logoff' period were sorted by occurrence time. we extracted only 'HH:mm' from the 'Date' field and subdivided it into 'Hour', '1-minute', '10-minutes', '15-minutes', and '30-minutes' fields, as shown in Table 8.

Table 8. Description of elements extracted for use as time feature factors

Field	Value	Description
Hour	00-23	24 Hour format
1-minute	00-59	1-minutes (from 0 to 59)
10-minute	00-05	10-minute intervals (6 intervals: 0 to 50)
15-minute	00-03	15-minute intervals (4 intervals: 0 to 45)
30-minute	00-01	30-minute intervals (2 intervals: 0 and 30)

We defined the field values for the 'Time' feature factor elements as follows:

- The 'Hour' field using a 24-hour format with integer values between 00 and 23.
- The '1-minute', '10-minute', '15-minute', and '30-minute' fields as integer values between 00 and K-1, where K is the quotient of 60 minutes divided by N minutes. N corresponds to the values (1, 10, 15, 30) that define the time feature coefficients.

In Step 2, we describe the process of constructing a model sequence using the four feature factor combination method. A model input sequence consists of a 'Location' feature factor token and a ('Behavior' feature factor + 'Time' feature factor) token. The 'Location' feature factor token corresponds to the value of the 'PC' field, which exists on the same row as the 'Logon' action in the 'Activity' field. The 'Behavior' feature factor token corresponds to the value of the 'Activity' field, and the 'Time' feature factor token corresponds to the combined value of the 'Time' feature factor elements. 'Time' feature factor elements refer to the ('Hour', '1-minute', '10-minute', '15-minute', '30-minute') fields created in Step 1. We propose four feature factor combination methods to provide ('Behavior' feature factor + 'Time' feature factor) as a follow-up token.

The following example explains how to combine the 'Behavior' feature factor token + 'Time' feature factor token:

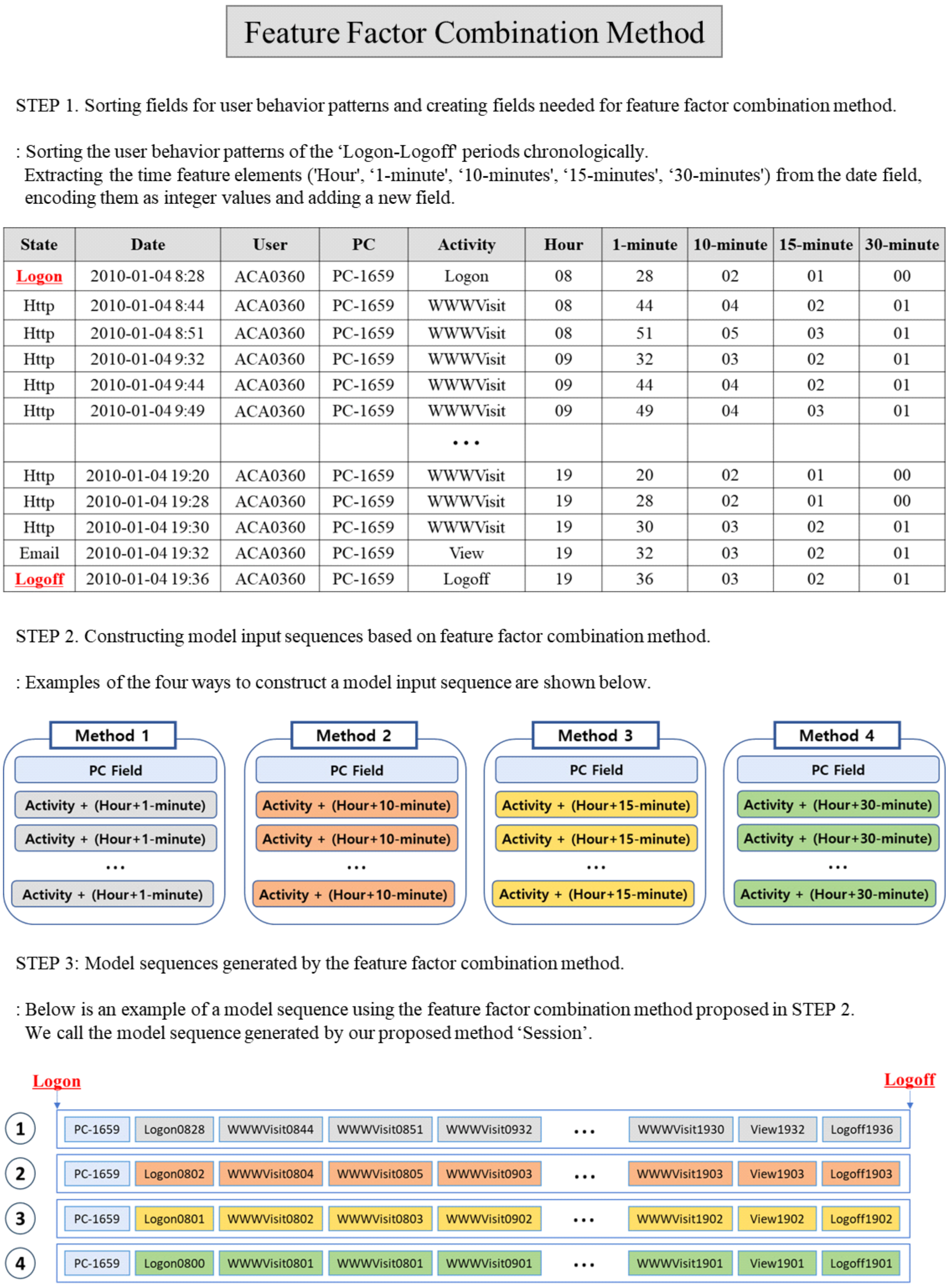
1. Extracting the 'Activity' field value ('Behavior' feature factor) and the 'Hour', '1-minute', '10-minute', '15-minute' and '30-minute' field values ('Time' feature factor elements) from the user's behavior pattern during the 'Logon-Logoff' period.
2. Creating one significant time feature factor by combining the 'Hour' field and the ('1-minute', '10-minute', '15-minute' and '30-minute') field.
3. Combining the 'Behavior' feature factor with + each 'Time' feature factor to create a new token.
4. Appending the created tokens after the 'Location' feature tokens to form a model sequence.

This process constructs a new model input sequence that contains the user's 'Time', 'Location', and 'Behavior' information. we define a sequence of model inputs created by this method of combining feature factors as a 'Session'.

In Step 3, we provide an example of building session data using each feature combination method. Session data 1, 2, 3, and 4 in Step 3 are the results of applying methods 1, 2, 3, and 4 in Step 2. The session data created by the feature factor combination method goes through feature preprocessing and vectorization processes to be used as model input data.

3.4 Feature Preprocessing and Vectorization

Figure 3 illustrates the steps involved in feature preprocessing and vectorization to prepare session data for input into the model.



Feature Factor Combination Method

STEP 1. Sorting fields for user behavior patterns and creating fields needed for feature factor combination method.

: Sorting the user behavior patterns of the 'Logon-Logoff' periods chronologically.
Extracting the time feature elements ('Hour', '1-minute', '10-minutes', '15-minutes', '30-minutes') from the date field, encoding them as integer values and adding a new field.

State	Date	User	PC	Activity	Hour	1-minute	10-minute	15-minute	30-minute
Logon	2010-01-04 8:28	ACA0360	PC-1659	Logon	08	28	02	01	00
Http	2010-01-04 8:44	ACA0360	PC-1659	WWWVisit	08	44	04	02	01
Http	2010-01-04 8:51	ACA0360	PC-1659	WWWVisit	08	51	05	03	01
Http	2010-01-04 9:32	ACA0360	PC-1659	WWWVisit	09	32	03	02	01
Http	2010-01-04 9:44	ACA0360	PC-1659	WWWVisit	09	44	04	02	01
Http	2010-01-04 9:49	ACA0360	PC-1659	WWWVisit	09	49	04	03	01
				···					
Http	2010-01-04 19:20	ACA0360	PC-1659	WWWVisit	19	20	02	01	00
Http	2010-01-04 19:28	ACA0360	PC-1659	WWWVisit	19	28	02	01	00
Http	2010-01-04 19:30	ACA0360	PC-1659	WWWVisit	19	30	03	02	01
Email	2010-01-04 19:32	ACA0360	PC-1659	View	19	32	03	02	01
Logoff	2010-01-04 19:36	ACA0360	PC-1659	Logoff	19	36	03	02	01

STEP 2. Constructing model input sequences based on feature factor combination method.

: Examples of the four ways to construct a model input sequence are shown below.

Method 1	Method 2	Method 3	Method 4
PC Field	PC Field	PC Field	PC Field
Activity + (Hour+1-minute)	Activity + (Hour+10-minute)	Activity + (Hour+15-minute)	Activity + (Hour+30-minute)
Activity + (Hour+1-minute)	Activity + (Hour+10-minute)	Activity + (Hour+15-minute)	Activity + (Hour+30-minute)
···	···	···	···
Activity + (Hour+1-minute)	Activity + (Hour+10-minute)	Activity + (Hour+15-minute)	Activity + (Hour+30-minute)

STEP 3: Model sequences generated by the feature factor combination method.

: Below is an example of a model sequence using the feature factor combination method proposed in STEP 2.
We call the model sequence generated by our proposed method 'Session'.

Logon → Logoff

①	PC-1659	Logon0828	WWWVisit0844	WWWVisit0851	WWWVisit0932	···	WWWVisit1930	View1932	Logoff1936
②	PC-1659	Logon0802	WWWVisit0804	WWWVisit0805	WWWVisit0903	···	WWWVisit1903	View1903	Logoff1903
③	PC-1659	Logon0801	WWWVisit0802	WWWVisit0803	WWWVisit0902	···	WWWVisit1902	View1902	Logoff1902
④	PC-1659	Logon0800	WWWVisit0801	WWWVisit0801	WWWVisit0901	···	WWWVisit1901	View1901	Logoff1901

Fig. 2. Feature factor combination method

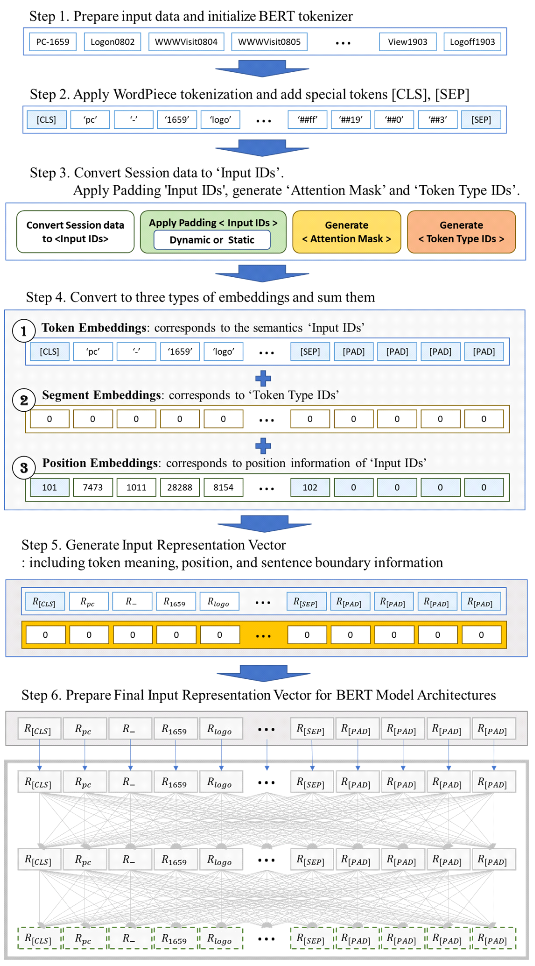

Step 1. Prepare input data and initialize BERT tokenizer
PC-1659
Logon0802
WWWVisit0804
WWWVisit0805
View1903
Logoff1903
Step 2. Apply WordPiece tokenization and add special tokens [CLS], [SEP]
[CLS]
'pc'
'-'
'1659'
'logo'
'##ff'
'##19'
'##0'
'##3'
[SEP]
Step 3. Convert Session data to 'Input IDs'. Apply Padding 'Input IDs', generate 'Attention Mask' and 'Token Type IDs'.
Convert Session data to <Input IDs>
Apply Padding < Input IDs >
Dynamic or Static
Generate < Attention Mask >
Generate < Token Type IDs >
Step 4. Convert to three types of embeddings and sum them
Token Embeddings: corresponds to the semantics 'Input IDs'
Segment Embeddings: corresponds to 'Token Type IDs'
Position Embeddings: corresponds to position information of 'Input IDs'
101
7473
1011
28288
8154
102
[PAD]
Step 5. Generate Input Representation Vector
: including token meaning, position, and sentence boundary information
Step 6. Prepare Final Input Representation Vector for BERT Model Architectures
$R_{[CLS]}$
R_{pc}
R_{-}
R_{1659}
R_{logo}
$R_{[SEP]}$
$R_{[PAD]}$

Fig. 3. Feature preprocessing and vectorization

In Step 1, we prepare the user behavior session data for input to BERT model and initialize BERT tokenizer. To tokenize the session data, we use the byte-pair encoding (BPE)-based 'WordPiece' tokenizer[40] provided by the HuggingFace Transformer library API.

In Step 2, we apply 'WordPiece' tokenization to the input session data. 'Wordpiece' tokenization is a method of splitting words into smaller sub-words. BERT model uses special tokens ([CLS], [SEP], etc.) to identify sentence structures and prefixes. The '[CLS]' (classification) token summarizes or categorizes information in an entire sequence, and the '[SEP]' (separator) token identifies boundaries between sentences or segments.

In Step 3, we convert the tokenized input session data into 'Input IDs'. Next, we apply a padding technique to the input IDs and generate an 'Attention Mask' and 'Token Type IDs'. 'Input IDs' result from converting each token in the tokenized input session data into a unique integer. We use dynamic and static padding techniques. Dynamic padding pads variable-length 'Input IDs' based on the batch's 'Input IDs' length. Static padding pads all 'Input IDs' to a pre-defined maximum length. Next, create an 'Attention Mask' of the 'Input IDs'. The 'Attention Mask' separates real tokens (1) from padding tokens (0) in the 'Input IDs'. Finally, create 'Token Type IDs' for the 'Input IDs'. The 'Token Type IDs' value is zero in single sentences. In pairs of sentences, the first sentence has a 'Token Type IDs' value of 0, and the second sentence has a 'Token Type IDs' value of 1. We utilize 'Token Type IDs' to prevent two logon-logoff periods in one input session data.

In step 4, we create three types of embeddings ('Token', 'Position', and 'Segment' embeddings) and sum them to make a single input representation vector. Token embeddings and Position embeddings use 'Input IDs', and Segment embeddings use 'Token Type IDs'. Token embeddings represent the meaning of each word or subword (token) in a vector space. Token embeddings capture the semantic characteristics of each token and allow semantic relationships to be learned by representing words with similar meanings with similar vectors. Position embeddings provide information about the position of each token in the sequence. Position embeddings learn order dependencies in language, allowing the same word to have different meanings or roles depending on its position in a sentence. Segment embedding utilizes 'Token Type IDs' to separate sentences. It recognizes the boundaries of each sentence and learns the relationships between sentences.

In step 5, the input representation vector (the sum of the three embeddings) and the 'Attention Mask' are combined to form the final representation vector that is the input to BERT model.

In Step 6, we provide the final representation vector as input to BERT model architecture to perform pre-training and transfer learning.

3.5 Pre-Training and Fine-Tuning

3.5.1 BERT Model Architectures

Bidirectional Encoder Representations from Transformers (BERT) model is a natural language processing (NLP) model that performs various tasks. BERT model catches complex semantic relationships, understands context effectively, and shows remarkable ability in representation learning. In addition, it can be pre-trained with large amounts of data, enabling effective transfer learning with only a small amount of data. In this paper, we use BERT Model Architecture as a pre-training model.

To determine the base model of the anomaly detection system, we apply various feature factor combination methods and feature preprocessing methods for each BERT model structure (Base, Medium, Small, Mini and Tiny). Table 9 describes BERT model architecture by hidden layer (L) and hidden embedding (H) size.

Table 9. BERT model names by architecture based on hidden layer (L) and hidden embeddings (H)

Hidden Layer (L)	Hidden Embeddings (H)			
	128	256	512	768
2	BERT-Tiny			
4		BERT-Mini	BERT-Small	
6				
8			BERT-Medium	
10				
12				BERT-Base

3.5.2 Pre-Training

Generally, BERT model is pre-trained using a masked language model (MLM) task and a next sentence prediction (NSP) task. MLM task involves training the model by randomly masking some tokens in the input data and then inferring the masked words.

NSP task focuses on identifying the relationship between sentences. Two sentences are provided as input data, and the model is trained to predict whether the second sentence follows the first sentence.

Recently, BERT variants have employed a variety of tasks to train more effectively in context. A Lite BERT (ALBERT)[41] is a lightweight version of BERT developed by Google and the Toyota Technological Institute in Chicago. It aims to reduce model size and improve learning speed while maintaining BERT's performance. Of particular note is that ALBERT replaces the traditional NSP task, inefficient for modeling sentence coherence, with the sentence order prediction (SOP) task. Decoding-enhanced BERT with Disentangled Attention (DeBERTa)[42] is a model developed by Microsoft that aims to improve the attention mechanism of BERT. Specifically, DeBERTa abandons the NSP operation and instead uses a separate attention mechanism and an improved mask decoder to handle content and location information better, improving the model's efficiency and overall performance. Efficiently Learning an Encoder that Classifies Token Replacements Accurately (ELECTRA)[43] is a model developed by Stanford University and Google that aims to improve how BERT models the mask language to increase the efficiency of dictionary learning. It introduces a replacement token detection (RTD) task instead of NSP. Robustly Optimized BERT Approach (RoBERTa)[44] is a model developed by Facebook AI Research that aims to optimize BERT's pre-training process. It determines that NSP operations are inefficient for downstream tasks and removes them entirely. Instead, it uses only MLM tasks with dynamic masking, significantly improving BERT's performance. XLNet[45] is a model developed by Carnegie Mellon University and Google that aims to overcome BERT's limitations. XLNet completely eliminates NSP and instead introduces a new approach called permutation language modeling (PLM). This method captures bi-directional context more effectively than NSP, addressing the mismatch between dictionary learning and fine-tuning caused by BERT's [MASK] token.

In this paper, we only use MLM tasks as a pre-training method for a user behavior analysis based anomaly detection system, according to recent research trends and a survey of various pre-training strategies. We use dynamic masking to prevent the pre-trained model from overfitting to user behavior patterns and to improve its generalization ability. As

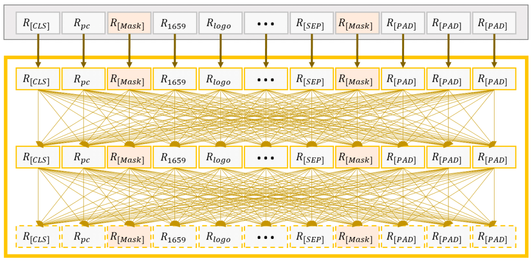


Fig. 4. BERT model architecture MLM tasks in the pre-training

the original BERT paper recommended, we randomly select 15% of all tokens. Of the selected tokens, 80% were replaced with the [MASK] special token, 10% with random tokens, and the remaining 10% unchanged. Figure 4 visualizes BERT model's architecture with MLM tasks in pre-training, and the [MASK] token is colored pink.

We exclusively use normal behavior session data for pre-training, and evaluate the MLM task using a separate set of normal behavior session data that was not used during the pre-training phase for validation.

Table 10 describes the common hyperparameters of BERT model architecture used for pre-training. In this paper, we set the following hyperparameters to optimize the model's performance: batch size 512 and learning rate 1e-5 for pre-training; dropout ratio 0.3 to avoid overfitting; AdamW optimizer (weight attenuation: 0.01) for the model's generalization performance; L1 regularization value set to (λ=1e-6) to reduce unnecessary noise by activating only some features and improve performance. We also applied gradient clipping (max_norm=0.5) for stable training of the model.

Table 10. Pre-training common hyperparameters

Hyperparameter	Values
Batch size	512
Max length	512
Learning rate	1e-5
Optimizer	AdamW
Weight decay	0.01
L1 regularization strength(λ)	1e-6
Gradient clipping max_norm	0.5
Dropout	0.3

3.5.3 Fine-Tuning

Fine-Tuning method is a type of transfer learning. It achieves effective results with limited data by using knowledge from pre-trained models. In this study, we employed BERT's fine-tuning approach to efficiently learn from a small amount of outlier data, using a 3:1 split ratio of normal to abnormal session data for model input. We attached a binary classifier[46] to the model's final layer for anomaly detection, utilizing the [CLS] token representation for downstream tasks. While maintaining most hyperparameters from pre-training (learning rate 1e-5, AdamW optimizer, weight decay 0.01, L1 regularization λ=1e-6, gradient clipping max_norm=0.5), we adjusted the batch size to 64 and increased dropout rate to 0.4 to prevent overfitting given the smaller dataset. CrossEntropyLoss was used as the loss function. Figure 5 illustrates BERT model architecture for classification during fine-tuning, while Table 11 details the complete hyperparameter configuration used in transfer learning.

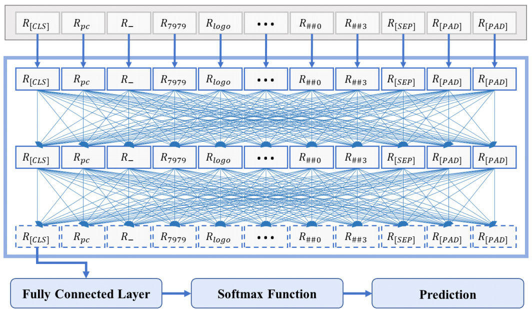


Fig. 5. BERT model architectures in the fine-tuning

Table 11. Transfer learning common hyperparameters

Hyperparameter	Values
Batch size	64
Max length	512
Learning rate	1e-5
Optimizer	AdamW
Weight decay	0.01
L1 regularization strength (λ)	1e-6
Gradient clipping max_norm	0.5
Dropout	0.4
Loss Function	nn.CrossEntropyLoss()

3.6 Performance Evaluation of the Anomaly Detection System

In this paper, we use the confusion matrix evaluation metric to evaluate the performance of a user behavior analysis based anomaly detection system.

3.6.1 Confusion Matrix Based Performance Metrics

Confusion matrix[47] is a metric primarily used to evaluate the predictive performance of binary classification models. It is often used to recognize specific types of errors or to evaluate the performance of unbalanced datasets[48]. Confusion matrix is a 2x2 matrix structure, and the performance metric consists of four combinations of actual and predicted classes. To comprehensively evaluate the reliability and efficiency of the anomaly detection system, we use Accuracy, Precision, Recall, F1-Score, and AUC-ROC[49]. Table 12 describes the four combinations of the confusion matrix, and Figure 6 shows the performance metrics according to confusion matrix combination method.

Table 12. Confusion matrix elements

Matrix Components	Description
True Positive (TP)	Correctly detected abnormal behavior as abnormal
False Positive (FP)	Incorrectly detected normal behavior as abnormal
False Negative (FN)	Incorrectly classified abnormal behavior as normal
True Negative (TN)	Correctly classified normal behavior as normal

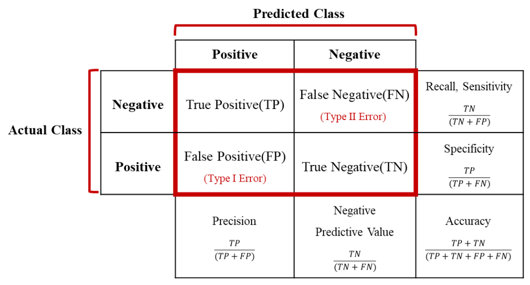


Fig. 6. Confusion matrix

Ⅳ. Experimental Results

4.1 Experimental Setup

Table 13 describes the configuration of the experimental environment for an anomaly detection system based on user behavior analysis.

Table 13. Experimental environment

Environment	Description	Ref.
OS	Ubuntu 22.04.2 LTS	
CPU	Intel(R) Xeon(R) Platinum 8480+ 48Core	
GPU	H100 160GB	
RAM	480GB	
python	3.10.6	50
pytorch	2.1.0	51
pandas	1.5.2	52
scikit-learn	1.2.0	53
transformers	4.34.1	32,54

4.2 Experimental Dataset

In the experiment, we perform EDA to identify user behavior patterns on the CERT r6.2 dataset of about 130 million data in its original raw data format. We reconstruct user behavior patterns in the EDA process and apply a feature factor combination method. Thus, we finally perform feature preprocessing and generate data of approximately 1.96 million sessions. Approximately 1.17 million (60%) of the session data was used to pre-train the models, and some of the remaining normal session data was used to validate the pre-trained models on the MLM task. We use both normal and abnormal session data for fine-tuning and system evaluation. Fine-tuning uses a small amount of 404 session data compared to the amount of session data used for pre-training. Table 14 describes the number of normal session data used for pre-training

Table 14. Number of pre-training and transfer learning sessions data

Pre-Training	Fine-Tuning		
	Train	Validation	Test
1,178,278	258	65	81

and the number of session data used for transfer learning.

4.3 Experimental Results and Analysis

In chapter 4.3.1, we check the feature combination methods and padding techniques that show high accuracy in MLM task based on pre-training results.

In chapter 4.3.2, we present the results of transfer learning for each BERT-Model architecture by applying the feature combination method and padding technique with the best performance in MLM task.

4.3.1 Evaluation of Pre-Trained Models Based on Feature Factor Combination Methods and Padding Techniques

To find the optimal balance between appropriate epochs and overfitting, we set the stabilization point of the pre-training model as follows:

(1) When the performance change of accuracy is 0.5% compared to the previous checkpoint

(2) When performance continues to improve, but the amount of increase decreases significantly

We pre-train 1-6 epochs for each BERT model structure to avoid overfitting and decreasing generalization ability. As a result, we find most BERT model architectures stabilize with pre-trained models in epochs 2-3. Table 15 indicates the pre-trained model's performance on the MLM Task according to the feature factor combination method and padding technique. BERT model architectures show higher accuracy when 'Behavior' feature factors and '(10-minute) Time' feature factors are combined, and static padding is applied.

4.3.2 Performance Evaluation by BERT Model Architecture

Table 16 shows the results of the performance evaluation of anomaly detection systems with the optimal feature factor combination method and padding technique for each BERT model architecture.

The performance evaluation shows that BERT-medium-uncased model performs the best, with an accuracy of 96.30%, followed by BERT-small-uncased model and BERT-mini-uncased model, which have the same accuracy of 95.06%. BERT-medium-uncased model also performs the best on F1-Score, an important performance metric for anomaly detection systems. We verify that BERT-Medium-uncased achieves a performance of 96.30% Accuracy, 96.27% Precision, 96.30% Recall, 96.28% F1-Score, and 99.10% AUC-ROC, showing the best-balanced performance numerically compared to the others of BERT Model architectures.

In this anomaly detection system, BERT-Medium-uncased model with a medium-sized structure achieves the highest performance when combining the 'Behavior' feature factor and the 'Hour + (10-minute) Time' feature factor and applying static padding. We found that it is possible to implement an anomaly detection system with sufficient dataset EDA, feature engineering, and feature preprocessing

Table 15. Pre-training model MLM task results based on feature factor combination methods and padding techniques

Model	Padding	1-minute	10-minute	15-minute	30-minute
BERT-Base-uncased	Dynamic	0.8014	0.8014	0.8015	0.8019
	Static	0.9381	0.9489	0.9083	0.8540
BERT-Medium-uncased	Dynamic	0.8011	0.8014	0.8013	0.8014
	Static	0.9381	0.9490	0.9099	0.8528
BERT-Small-uncased	Dynamic	0.8011	0.8012	0.8013	0.8013
	Static	0.9328	0.9480	0.9087	0.8471
BERT-Mini-uncased	Dynamic	0.8011	0.8012	0.8011	0.8011
	Static	0.9288	0.9460	0.9054	0.8477
BERT-Tiny-uncased	Dynamic	0.8010	0.8010	0.8010	0.8011
	Static	0.8858	0.9317	0.8961	0.7461

Table 16. Performance evaluation metrics for pre-trained models with optimal feature factor combination method and padding techniques

Model	Accuracy	Precision	Recall	F1-Score	AUC-ROC
BERT-Base uncased	0.9012	0.8991	0.9012	0.8994	0.9000
BERT-Medium uncased	0.9630	0.9627	0.9630	0.9626	0.9910
BERT-Small uncased	0.9506	0.9537	0.9506	0.9487	0.9295
BERT-Mini uncased	0.9506	0.9536	0.9506	0.9514	0.9475
BERT-Tiny uncased	0.7531	0.5671	0.7531	0.6470	0.7885

without using pre-trained models with complex structures.

Ⅴ. Conclusion

In this paper, we applied feature factor combination and preprocessing methods to user behavior data to construct suitable input data for the model. We aimed to implement an anomaly detection system using an optimally scaled BERT model.

We proposed a feature factor combination method and preprocessing technique to convert user behavior data into input data of BERT model. We conducted an ablation study to find the optimal scale model structure. We found that token construction is very important for effectively learning user behavior patterns. In particular, the combination of (time + behavior) feature factors and the input sequence padding method significantly impacted the model's performance. Static padding allowed us to keep the input sequence length constant while preserving the temporal information, which we found effective for learning behavior patterns. In the experimental results, we analyzed the performance of each time step. We found that '1-minute' time features factor showed low performance due to increased complexity and noise caused by excessive segmentation. '30-minute' time feature factor showed low performance because the time interval is too large, causing important behavioral information to be lost. On the other hand, we can see that the best performance is achieved by BERT-Medium-uncased model, which has an F1 Score of 96.28% and an AUC-ROC of 99.10% for the '10-minutes' time feature factor. The results suggest that proper segmentation and padding of the data significantly impact the model's performance for behavioral pattern analysis.

In future work, we plan to extend the anomaly detection system to include various types of user behavior data, such as network traffic datasets, to be applied in real-world environments. Additionally, we aim to apply BERT Variants and other natural language processing models to user anomaly detection. Furthermore, we intend to improve the system's detection performance by developing the feature factor combination method proposed in this paper.

References

[1] Verizon, *2024 Data Breach Investigations Report* (2024), Retrieved Sep. 9, 2024, from https://www.verizon.com/business/resources/reports/dbir/

[2] Proofpoint, *Voice of the CISO Report* (2024), Retrieved Sep. 9, 2024, from https://www.proofpoint.com/kr/resources/white-papers/voice-of-the-ciso-report.

[3] N. R. Prasad, S. Almanza-Garcia, and T. T. Lu, "Anomaly detection," *Comput. Mater. Contin.*, vol. 14, no. 1, pp. 1-22, 2009. (https://doi.org/10.3970/cmc.2009.014.001)

[4] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, Article 15, pp. 1-58, Jul. 2009. (https://doi.org/10.1145/1541880.1541882)

[5] S. Omar, A. Ngadi, and H. H. Jebur, "Machine learning techniques for anomaly detection: An overview," *Int. J. Comput. Appl.*, vol. 79, no. 2, pp. 33-41, Oct. 2013. (https://doi.org/10.5120/13715-1478)

[6] A. B. Nassif, M. A. Talib, Q. Nasir, and F. M. Dakalbab, "Machine learning for anomaly detection: A systematic review," *IEEE Access*, vol. 9, pp. 78658-78700, May 2021. (https://doi.org/10.1109/ACCESS.2021.3083060)

[7] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A Survey," *arXiv preprint arXiv:1901.03407*, Jan. 2019. Retrieved Sep. 09, 2024, from https://arxiv.org/abs/1901.03407.
(https://doi.org/10.48550/arXiv.1901.03407)

[8] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, vol. 54, no. 2, Article 38, pp. 1-38, Mar. 2021.
(https://doi.org/10.1145/3439950)

[9] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural Computation*, vol. 12, no. 5, pp. 1207-1245, May 2000.
(https://doi.org/10.1162/089976600300015565)

[10] K. Yang, S. Kpotufe, and N. Feamster, "An efficient one-class SVM for anomaly detection in the internet of things," *arXiv preprint arXiv:2104.11146*, Apr. 2021. Retrieved Sep. 09, 2024, from https://arxiv.org/abs/2104.11146
(https://doi.org/10.48550/arXiv.2104.11146)

[11] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly detection using one-class neural networks," *arXiv preprint arXiv:1802.06360*, Jan. 2019. Retrieved Sep. 09, 2024, from https://arxiv.org/abs/1802.06360.
(https://doi.org/10.48550/arXiv.1802.06360)

[12] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognition*, vol. 58, pp. 121-134, Oct. 2016.
(https://doi.org/10.1016/j.patcog.2016.03.028)

[13] A. Binbusayyis and T. Vaiyapuri, "Unsupervised deep learning approach for network intrusion detection combining convolutional autoencoder and one-class SVM," *Applied Intell.*, vol. 51, pp. 7094-7108, Oct. 2021.
(https://doi.org/10.1007/s10489-021-02205-9)

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proc. IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.
(https://doi.org/10.1109/5.726791)

[15] D. Kwon, K. Natarajan, S. C. Suh, H. Kim, and J. Kim, "An empirical study on network anomaly detection using convolutional neural networks," in *Proc. IEEE 38th ICDCS 2018*, pp. 1595-1598, Vienna, Austria, Jun. 2018.
(https://doi.org/10.1109/ICDCS.2018.00178)

[16] M. K. Hooshmand and D. Hosahalli, "Network anomaly detection using deep learning techniques," *CAAI Trans. Intell. Technol.*, vol. 7, no. 2, pp. 228-243, Jun. 2022.
(https://doi.org/10.1049/cit2.12078)

[17] H.-J. Im, T.-R. Kim, J.-G. Song, and B.-S. Kim, "Anomaly detections model of aviation system by CNN," *J. Aerospace Syst. Eng.*, vol. 17, no. 4, pp. 67-74, Aug. 2023.
(https://doi.org/10.20910/JASE.2023.17.4.67)

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735-1780, Nov. 1997.
(https://doi.org/10.1162/neco.1997.9.8.1735)

[19] B. Lindemann, B. Maschler, N. Sahlab, and M. Weyrich, "A survey on anomaly detection for technical systems using LSTM networks," *Comput. in Industry*, vol. 131, no. 103498, Jun. 2021.
(https://doi.org/10.1016/j.compind.2021.103498)

[20] M. S. Elsayed, N. Le-Khac, S. Dev, and A. D. Jurcut, "Network anomaly detection using LSTM based autoencoder," in *Proc. 16th ACM Symp. QoS and Secur. for Wireless and Mob. Netw. (Q2SWinet '20)*, pp. 37-45, New York, NY, USA, Nov. 2020.
(https://doi.org/10.1145/3416013.3426457)

[21] T. Ergen and S. S. Kozat, "Unsupervised anomaly detection with LSTM neural networks," *IEEE Trans. Neural Netw. and Learn. Syst.*, vol. 31, no. 8, pp. 3127-3141, Aug. 2020.
(https://doi.org/10.1109/TNNLS.2019.2935975)

[22] T. Kim and S. Cho, "Web traffic anomaly detection using C-LSTM neural networks," *Expert Syst. with Appl.*, vol. 106, pp. 66-76, Apr. 2018.

(https://doi.org/10.1016/j.eswa.2018.04.004)

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Proc.: Explorations in the Microstructures of Cognition*, vol. I, pp. 318-362, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986. (https://doi.org/10.7551/mitpress/4943.003.0128)

[24] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proc. 23rd ACM SIGKDD Int. Conf. KDD '17*, pp. 665-674, New York, NY, USA, Aug. 2017. (https://doi.org/10.1145/3097983.3098052)

[25] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau, "Autoencoder-based network anomaly detection," in *Proc. 2018 WTS*, pp. 1-5, Phoenix, AZ, USA, Apr. 2018. (https://doi.org/10.1109/WTS.2018.8363930)

[26] R. C. Aygun and A. G. Yavuz, "Network anomaly detection with stochastically improved autoencoder based models," in *Proc. 2017 IEEE 4th Int. Conf. Cyber Security and Cloud Comput. (CSCloud)*, pp. 193-198, New York, NY, USA, Jun. 2017. (https://doi.org/10.1109/CSCloud.2017.39)

[27] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proc. 2nd ICLR 2014*, pp. 1-14, Banff, Canada, Apr. 2014. (https://doi.org/10.48550/arXiv.1312.6114)

[28] T. Iqbal and S. Qureshi, "Reconstruction probability-based anomaly detection using variational auto-encoders," *Int. J. Comput. and Appl.*, vol. 45, no. 3, pp. 231-237, Nov. 2022. (https://doi.org/10.1080/1206212X.2022.2143026)

[29] D. Zimmerer, S. A. A. Kohl, J. Petersen, F. Isensee, and K. H. Maier-Hein, "Context-encoding variational autoencoder for unsupervised anomaly detection," in *Proc. Int. Conf. MIDL 2019*, pp. 1-8, London, UK, Jul. 2019. (https://doi.org/10.48550/arXiv.1812.05941)

[30] J. Sun, X. Wang, N. Xiong, and J. Shao, "Learning sparse representation with variational auto-encoder for anomaly detection," *IEEE Access*, vol. 6, pp. 33353-33361, Jun. 2018. (https://doi.org/10.1109/ACCESS.2018.2848210)

[31] R. Yao, C. Liu, L. Zhang, and P. Peng, "Unsupervised anomaly detection using variational auto-encoder based feature extraction," in *Proc. 2019 IEEE Int. Conf. Prognostics and Health Manag. (ICPHM)*, pp. 1-7, San Francisco, CA, USA, Jun. 2019. (https://doi.org/10.1109/ICPHM.2019.8819434)

[32] 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. Advances in NIPS 2017*, vol. 30, pp. 5998-6008, Long Beach, USA, Dec. 2017. (https://doi.org/10.48550/arXiv.1706.03762)

[33] J. Devlin, M. 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 Assoc. Computat. Linguistics: Human Language Technol. (NAACL-HLT 2019)*, pp. 4171-4186, Minneapolis, USA, Jun. 2019. (https://doi.org/10.48550/arXiv.1810.04805)

[34] W. Dang, B. Zhou, L. Wei, W. Zhang, Z. Yang, and S. Hu, "TS-Bert: Time series anomaly detection via pre-training model Bert," in *Proc. Int. Conf. Database Syst. for Advanced Appl. (DASFAA 2021)*, pp. 220-235, Taipei, Taiwan, Apr. 2021. (https://doi.org/10.1007/978-3-030-77964-1_17)

[35] J. Guo, X. Chen, Z. Liu, Y. Chen, and C. Xu, "LogBERT: Log anomaly detection via BERT," in *Proc. 2021 IJCNN*, pp. 1-8, Shenzhen, China, Jul. 2021. (https://doi.org/10.1109/IJCNN52387.2021.9534113)

[36] X. Tang and Y. Guan, "Log anomaly detection based on BERT," *Signal, Image and Video Process.*, vol. 18, no. 2, pp. 1139-1147,

Feb. 2024.
(https://doi.org/10.1007/s11760-024-03327-6)

[37] B. Lindauer, *Insider Threat Test Dataset*(2020), Retrieved Sep. 9, 2024, from https://kilthub.cmu.edu/articles/dataset/Insider_Threat_Test_Dataset/12841247/1
(https://doi.org/10.1184/R1/12841247.v1.)

[38] J. W. Tukey, *Exploratory Data Analysis*(1977), Retrieved Sep. 9, 2024, from https://archive.org/details/exploratorydataa00tuke_0/page/n19/mode/2up.

[39] A. Gelman, "Exploratory data analysis for complex models," *J. Comput. and Graphical Statistics*, vol. 13, no. 4, pp. 755-779, Dec. 2004.
(https://doi.org/10.1198/106186004X11435)

[40] Y. Wu, M. Schuster, Z. Chen, et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint, vol. 1609.08144*, Oct. 2016. Retrieved Sep. 9, 2024, from https://arxiv.org/abs/1609.08144.
(https://doi.org/10.48550/arXiv.1609.08144)

[41] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," in *Proc. 8th ICLR 2020*, pp. 1-17, Addis Ababa, Ethiopia, Apr. 2020.
(https://doi.org/10.48550/arXiv.1909.11942)

[42] P. He, X. Liu, J. Gao, and W. Chen, "DeBERTa: Decoding-enhanced BERT with disentangled attention," in *Proc. 9th ICLR 2021*, pp. 1-21, Virtual Event, May 2021.
(https://doi.org/10.48550/arXiv.2006.03654)

[43] K. Clark, M. Luong, Q. V. Le, and C. D. Manning, "ELECTRA: Pre-training text encoders as discriminators rather than generators," in *Proc. 8th ICLR 2020*, pp. 1-18, Addis Ababa, Ethiopia, Apr. 2020.
(https://doi.org/10.48550/arXiv.2003.10555)

[44] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," *arXiv preprint arXiv:1907.11692*, Jul. 2019. Retrieved Sep. 9, 2024, from https://arxiv.org/abs/1907.11692.
(https://doi.org/10.48550/arXiv.1907.11692)

[45] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized autoregressive pretraining for language understanding," in *Proc. Advances in NeurIPS 2019*, vol. 32, pp. 5753-5763, Vancouver, Canada, Dec. 2019.
(https://doi.org/10.48550/arXiv.1906.08237)

[46] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune BERT for text classification?," in *Proc. China National Conf. Chinese Comput. Linguistics (CCL 2019)*, pp. 194-206, Kunming, China, Oct. 2019.
(https://doi.org/10.48550/arXiv.1905.05583)

[47] 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)

[48] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 20-29, Jun. 2004.
(https://doi.org/10.1145/1007730.1007735)

[49] 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)

[50] Python Software Foundation, *Python 3.10.6 Release*(2022), Retrieved Sep. 09, 2024, from https://www.python.org/downloads/release/python-3106/

[51] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Advances in NeurIPS 2019*, vol. 32, pp. 8024-8035, Vancouver, Canada, Dec. 2019.
(https://doi.org/10.48550/arXiv.1912.01703)

[52] W. McKinney, "Data structures for statistical computing in python," in *Proc. 9th Python in Sci. Conf.*, pp. 56-61, Austin, Texas, Jun. 2010. (https://doi.org/10.25080/Majora-92bf1922-00a)

[53] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825-2830, 2011. (https://www.jmlr.org/papers/v12/pedregosa11a.html)

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.

<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. 1997~Apr. 2001 : Researcher, Agency for Defense Development (ADD)

May. 2001~Current : Team leader/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]

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]