國立臺北科技大學

資訊安全學位學程

碩士學位論文

S2GE-NIDS: A hybrid architecture combining structured semantics and generation embedded network intrusion detection system in IoT

研究生：周玟萱

指導教授：陳香君博士

中華民國一百一十四年七月

國立臺北科技大學

資訊安全學位學程

碩士學位論文

S2GE-NIDS: A hybrid architecture combining structured semantics and generation embedded network intrusion detection system in IoT

研究生：周玟萱

指導教授：陳香君博士

中華民國一百一十四年七月

「學位論文口試委員會審定書」掃描檔

審定書填寫方式以系所規定為準，但檢附在電子論文內的掃描檔須具備以下條件：

1. 含指導教授、口試委員及系所主管的完整簽名。
2. 口試委員人數正確，碩士口試委員至少 3 人、博士口試委員至少 5 人。
3. 若此頁有論文題目，題目應和書背、封面、書名頁、摘要頁的題目相符。
4. 此頁有無浮水印皆可。

審定書不用頁碼

Abstract

Keyword: IoT Security, Information Security, Anomaly Detection, Multilayer Perceptron, Semantic Vector

As network environments become increasingly complex and dynamic, traditional intrusion detection methods struggle to keep pace with evolving threats and high-volume traffic. This paper proposes an efficient anomaly detection framework that leverages hash-based token embedding and a lightweight multi-layer perceptron (MLP) for the semantic representation of network flows. By transforming feature values into semantic tokens and utilizing a hashing trick for embedding lookup, our approach enables scalable and robust processing without maintaining an explicit vocabulary. The resulting embedding vectors are flattened and processed by the MLP to produce semantic vectors, which are clustered using a center loss strategy for unsupervised anomaly detection. Experimental results on public benchmark datasets demonstrate that our method achieves competitive accuracy with significantly improved computational efficiency compared to traditional attention-based models.

Acknowledgements

During this research, I sincerely thank my supervisor Professor Chen Xiangjun for her careful guidance and generous teachings. The professor not only gave me great inspiration and help in academics, but also provided valuable advice on research methods and paper structure, allowing me to continue to improve and make breakthroughs. In addition, the professor's rigorous academic attitude and extensive professional knowledge also deeply inspired me to pursue excellence. I thank the professor for his constant encouragement and support during the research, which enabled me to persevere in the face of challenges and successfully complete this paper. Without the professor's careful cultivation and guidance, this research would not have been able to achieve such results. I would like to express my most sincere gratitude and respect.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Driven by the rapid advancement of digital transformation and smart infrastructure, the Internet of Things (IoT) has emerged as a cornerstone of next-generation information technology. Through the integration of sensors, embedded devices, communication modules, and platform software, IoT enables physical objects to communicate in real time and generate massive volumes of data. These data streams support a broad range of applications, such as smart manufacturing, intelligent transportation, remote healthcare, and smart homes, yielding substantial economic and societal value [1].

However, as the number of connected devices increases and deployment scenarios become more complex, IoT systems face unprecedented cybersecurity challenges. Many IoT devices are resource-constrained, infrequently updated, and difficult for users to manage. With limited encryption and a lack of monitoring mechanisms, these devices become prime targets for cyber intrusions and attacks. Effectively identifying abnormal behaviors and hidden threats in IoT network traffic has therefore become a pressing research priority.

Furthermore, existing intrusion detection technologies often struggle to adapt to evolving threats. While deep learning approaches such as Word2Vec **sugathadasa2017synergistic** and Transformer-based models **han2022survey** have demonstrated semantic learning capabilities, they also introduce critical drawbacks: large vocabulary requirements, high computational complexity, and limited flexibility in dynamic or resource-constrained environments.

To address these limitations, we propose Structured Semantics and Generation Embedded Network Intrusion Detection System (S2GE-NIDS) a lightweight, interpretable anomaly detection framework designed for IoT environments. S2GE-NIDS combines hash-based token embedding with a multi-layer perceptron (MLP) model and introduces a linked-list mechanism to mitigate hash collisions inherent to non-cryptographic hash functions such as MurmurHash3 **yamaguchi2013hardware**. This design enables efficient feature encoding while avoiding the need to maintain a large vocabulary.

In our approach, network packets are first transformed into semantic tokens and encoded using hash-based indexing. The resulting embedding vectors are concatenated into a single, fixed-length semantic vector, which is processed by an MLP and projected near a learned semantic center. Any significant deviation from this center measured by Mahalanobis distance is classified as a potential anomaly [2].

The proposed S2GE-NIDS framework offers several advantages over conventional intrusion detection systems. First, it uses a hash-based embedding approach; The propose of the hash table in our method is efficiently searching data and storage and the combination of double hashing **guibas1976analysis** and linked list **dietz1982maintaining** serial nodes can compress and pre-allocate to save index space and reduce dynamic memory configuration costs. In addition, a fixed-size index after modulo operation is used to ensure that the size of the embedded table is controllable, taking into account both storage space and hashing uniformity. This strategy has better space complexity than directly hashing the entire key-value pair.

Second, the model provides a mathematically interpretable anomaly scoring mechanism by integrating Mahalanobis distance, which quantifies how far a sample deviates from the learned distribution of normal behavior. This not only improves detection accuracy but also enables explainable results.

Third, the system is lightweight and highly efficient, relying on simple MLP-based encoding instead of complex deep architectures **naveed2023comprehensive**, making it well-suited for deployment in real-time or resource-constrained environments such as edge devices in IoT networks. Therefore, this double hashing architecture not only effectively optimizes the hash function combination, makes the hash more uniform, and reduces the chain length. It also improves the stability and efficiency of hash embedding, laying a good foundation for the subsequent anomaly detection model based on semantic vectors.

The structure of this paper is organized as follows. Chapter 2 provides the background knowledge related to S2GE-NIDS. Chapter 3 presents the architecture and methodology of the proposed framework, detailing the design and each module. Chapter 4 describes the implementation setup and steps, Chapter 5 provides the experimental results. Finally, Chapter 6 concludes and outlines for future research.

Chapter 2 Related Work

This section will introduce the relevant basic knowledge, including the existing the IoT network intrusion detection methods, Tokenization, Hash Embedding, and Multi-Layer Perceptron,Semantic Vector and Mahanobis Distance.

2.1 Network Intrusion Detection System in IoT

In recent years, the proliferation of Internet of Things (IoT) devices has led to an increased focus on developing effective network intrusion detection systems (NIDS) tailored to the specific characteristics of IoT environments. Various approaches have been proposed to address the challenges associated with high-volume, heterogeneous network traffic, constrained device capabilities, and evolving attack patterns.

For example, Kharoubi et al. [3] proposed a convolutional neural network (CNN)-based detection system designed for IoT security. By applying CNN layers to extract spatial features from traffic data, the model achieved high classification performance on datasets such as CICIoT2023 [4] and CICIoMT2024 [5]. The authors demonstrated that their method achieved excellent precision and recall in both binary and multi-class scenarios. However, a notable limitation of the CNN-based approach lies in its inability to fully capture temporal dependencies across packet sequences, and its reliance on supervised learning requires extensive labeled datasets.

Ashraf et al. [6] introduced a Internet of Things Network Intrusion Detection System (INIDS) based on traditional machine learning classifiers applied to the BoT-IoT dataset. The study compared seven algorithms, including Random Forest, Artificial Neural Networks (ANN), and Support Vector Machines etc. Their results showed that Random Forest and ANN achieved the highest accuracy and robustness among all tested classifiers. Despite its efficiency, the NIDS system was highly dependent on manual feature engineering and lacked adaptability to novel threats, which are critical in fast-evolving IoT environments.

Lee et al. [7] proposed a method for extracting features from network traffic to build models that can effectively detect intrusions. They thus demonstrated that feature selection has a critical impact on the accuracy and efficiency of NIDS, especially when dealing with large datasets or new types of attacks.

Thaseen et al. [8] proposed a method combining feature selection with multi-class Support Vector Machine(SVM) to improve the accuracy of NIDS. They demonstrated that a good feature selection strategy can effectively reduce detection errors and improve classification efficiency.

Table 2.1 provides a consolidated overview of eight widely adopted features commonly utilized in anomaly detection across both academic research and industrial applications.

Table 2.1 Common Anomalous Features in IoT Network Traffic and Their Descriptions

Feature	**Description**
Destination Port [9]	The port number in a network packet received by the destination host, used to identify the communication entry point of an application or service.
Protocol Type [10]	Used to indicate the type of communication protocol used by the packet, such as TCP, UDP, ICMP, etc.
Duration / flow_duration [10]	The length of time a network connection session lasts, the number of seconds from the start to the end of the connection.
Packet Length [9]	The size of a packet is usually measured in bytes and refers to the amount of data in a single packet.
Source IP / Destination IP [11]	The source and destination IP addresses of a packet indicate the network locations of the sender and receiver, respectively.
Flow Bytes per Second [9]	The total data flow through the network connection per unit time, measured in bytes per second (Bytes/s).
TCP Flags [12]	The control bit flag in the TCP packet indicates the status of the packet or the control message, such as SYN for connection request and FIN for connection termination.
Number of Connections [9]	The number of network connections established by a single source IP within a specified period of time, used to measure connection activity.

2.2 Tokenization Technique in IoT Application

Tokenization is the process of converting raw packet data or traffic feature fields into semantically meaningful token sequences, thereby enabling anomaly detection models to perform contextual understanding and analysis. This technique facilitates the modeling of complex patterns in network traffic by translating low-level features into high-level representations.

Shapira et al. [13] proposed Flow2Vec, a framework that transforms network flow events into token sequences and applies contextual embeddings for analysis. This method is particularly effective for the classification and anomaly detection of encrypted traffic, as it captures the semantic relationships among protocols, IP addresses, and packet sizes. Li, Wei et al. [14] transformed URL paths and DNS queries in IoT traffic into text sequences. They performed n-gram segmentation, followed by TF-IDF or Word2Vec embedding, and combined these representations with SVM and RandomForest classifiers to detect malicious domains.

Karim et al. [15] introduced a technique that tokenizes IoT network traffic features—such as protocols, port numbers, and flag bits—and processes them through an embedding layer followed by a Long Short-Term Memory (LSTM) network for semantic modeling. This approach demonstrated high classification accuracy and recall in identifying IoT malware samples. Building on this idea, Muhammad et al. [16] proposed a lightweight method combining token embedding with a deep classification model. Their technique tokenizes and standardizes packet fields such as timestamps, lengths, and protocol names, yielding significant improvements in real-time classification performance and anomaly detection, especially in resource-constrained IoT environments.

Javaid et al. [17] employed both One-Hot encoding and word embedding for categorical features, such as protocol types and flag statuses, in IoT networks. These representations were input into deep neural networks to detect abnormal traffic. Experimental results demonstrated that embedding semantic information not only improves detection accuracy but also enhances generalization while reducing feature dimensionality.

Collectively, these studies confirm that tokenization strategies are highly effective in the context of IoT anomaly detection. By transforming heterogeneous traffic attributes into unified embedding vectors, such approaches enable models to learn and infer behavioral patterns across both packet-level and application-level traffic. This has significant implications for the scalability and accuracy of intrusion detection systems deployed in diverse and dynamic IoT environments.

Table 2.2 shows the comparison of some features in anomaly detection using Tokenization. For example, Protocol = TCP only retains the field name and value, and directly discards other symbols and spaces.

Table 2.2 Examples of Field-Value Tokenization in IoT Network Traffic

Feature Field	**Tokenized Representation**
Protocol = TCP	Protocol:TCP
Destination Port = 80	DstPort:80
Flow Duration = 0.32817	FlowDuration:0.32817
Source IP = 192.168.0.1	SrcIP:192.168.0.1
Payload Bytes = 56	PayloadBytes:56
Packet Count = 10	PacketCount:10
Flag = ACK	Flag:ACK
Destination IP = 10.0.0.5	DstIP:10.0.0.5

2.3 Hash Embedding in Anomaly Detection

Hash Embedding is a common lightweight feature encoding technology [18], which is particularly suitable for structured, high-dimensional, or large-number-of-categories network data. Its core approach is to convert each field name/field value (or a combination of the two) into a set of indexes through a hash function (such as MurmurHash3), and query the embedding table to obtain a fixed-length semantic vector. Ashraf et al.

As Gupta et al. [19] proposed a hash embedding-based method for representing protocol-level IoT traffic, especially targeting categorical fields such as destination ports, device types, and payload signatures. Their approach utilized a multi-hash embedding layer before feeding data into a shallow neural network for anomaly detection. Experiments on the IoT dataset showed a 40% reduction in model size while retaining over 97% detection accuracy compared to one-hot encoding.

Feng et al. [20] had further integrated hash embeddings into a lightweight convolutional architecture for edge-based IoT security. Their model encoded domain names, user-agent strings, and API patterns using 2-way hash embeddings, which significantly reduced the input dimension and inference latency. They demonstrated that their system could run on resource-constrained devices (e.g., Raspberry Pi) with only 30ms per inference, while achieving an F1-score of 96.5% on the CIC-ToN-IoT dataset.

Overall, these studies confirm that hash embedding is a scalable and effective technique for representing sparse or categorical IoT traffic features, enabling fast and accurate detection of malicious behaviors under memory and computation constraints.

2.4 Multi-Layer Perceptron in Anomaly Detection

Multi-Layer Perceptrons (MLPs) have been widely applied in the field of anomaly detection due to their capability to model non-linear relationships between input features and hidden patterns. Unlike traditional statistical models that rely on predefined thresholds or assumptions about data distribution, MLPs are capable of learning complex, high-dimensional feature representations in a data-driven manner [21].

In recent years, MLP-based anomaly detection methods have been employed in various domains, including network security [22], industrial control systems [23], and IoT environments [24]. These models typically consist of multiple fully connected layers with nonlinear activation functions, such as ReLU or sigmoid, enabling the learning of hierarchical semantic features. The outputs are used to distinguish between normal and abnormal behavior based on reconstruction error, classification scores, or learned distance metrics.

Moustafa et al. [22] proposed a hybrid intrusion detection system that combines feature selection and deep learning, utilizing a Multilayer Perceptron (MLP) as the final classifier. Experimental results demonstrated that the hybrid approach significantly outperforms classical machine learning algorithms such as Decision Trees and Naive Bayes, achieving over 95% detection accuracy and a low false positive rate, particularly excelling in identifying DoS and probe attacks.

Nguyen et al. [24] developed an anomaly detection method for IoT traffic using an autoencoder framework, with the decoder implemented as a Multilayer Perceptron. They focused on reducing communication overhead while maintaining detection accuracy, suitable for low-bandwidth IoT networks. The model takes raw traffic features (e.g., port numbers, packet sizes) and encodes them into a compact latent space before reconstructing them through a multi-layer MLP. Anomalies are identified based on high reconstruction error. Their experiments on the BoT-IoT dataset showed that the MLP-based decoder could detect attacks like DDoS and port scanning with an F1-score exceeding 98.5%, while maintaining a false positive rate below 1%, thus demonstrating the effectiveness of MLP in semantic compression and inference within constrained IoT devices.

Nathan Shone et al. [25] introduced a hybrid deep learning approach combining a stacked autoencoder with an MLP classifier to detect network intrusions. Their model was evaluated on the NSL-KDD dataset, achieving an accuracy of 85.42% and demonstrating superior performance over classical ML algorithms such as decision trees and SVM.

Similarly et al. [26] applied a pure MLP-based architecture for anomaly detection in the BoT-IoT dataset. The network consisted of three hidden layers with ReLU activation and dropout regularization. The results showed that MLP achieved over 98.5% detection accuracy and maintained a false positive rate below 1%, outperforming traditional algorithms such as KNN and Naive Bayes.

Ahmad Javaid et al. [17] further explored MLP in a deep learning pipeline tailored for IoT environments. They emphasized the importance of feature normalization and used a softmax output layer for multi-class classification. Their experiments on KDDCup'99 and UNSW-NB15 datasets revealed that MLP models trained on optimized features could achieve both high recall and precision in detecting diverse attack types, including DoS, probing, and user-to-root exploits.

These findings suggest that MLP can serve as a strong baseline model in IoT anomaly detection pipelines, especially when combined with proper feature engineering and regularization techniques.

2.5 Semantic Vector in IoT Anomaly Detection

Semantic vector representations, originally popularized in natural language processing (NLP), have gained traction in anomaly detection tasks due to their ability to encode complex contextual information into fixed-length embeddings. In security-related applications, raw network traffic often contains heterogeneous features that lack explicit semantics; transforming these into semantic vectors enables better generalization and interpretability [27].

Recent works have applied semantic encoding strategies, such as Word2Vec and sequence embeddings, to convert protocol names, IP addresses, or header fields into high-dimensional vectors [14]. These semantic vectors capture latent relationships between fields and behaviors, allowing downstream models to detect subtle deviations from normal patterns. For instance, Shapira et al. [13] proposed Flow2Vec, which encodes sequences of network events into dense vectors, improving anomaly detection in encrypted traffic.

Torres et al. [28] used a self-supervised Transformer model to learn semantic embeddings of packet sequences, tokenized each field and value and converted them into word embeddings, and finally achieved anomaly classification accuracy of over 98% on the TON IoT dataset.

Rahman et al. [29] treated DNS/URL traffic as a text sequence, constructed semantic vectors using n-gram segmentation and TF-IDF, and then used SVM and Random Forest for classification, with an F1-score of over 96% for detecting malicious domains.

Nguyen et al. [30] concatenated the structured fields of IoT packets through semantic embedding and entered the AutoEncoder model for reconstruction error analysis. The study pointed out that compared with pure numerical encoding, semantic vectors can effectively improve anomaly recall and precision.

2.6 Mahanobis Distance in IoT Anomaly Detection

Mahalanobis distance was first proposed by Indian statistician Prasanta Chandra Mahalanobis. It proposed a method to measure the ”distance” between points and multidimensional statistical distributions, thus breaking through the limitation of Euclidean distance that cannot adjust scale and correlation. Venturini et al. [31] explored the application of Mahalanobis distance in smart home behavior analysis, using multidimensional time series to capture abnormal device usage scenarios. Experiments show that when Mahalanobis distance exceeds the normal threshold, abnormal behaviors such as failures or unexpected operations can be detected. The proposed of following classic formula as below.

$$d_M(\mathbf{x}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \, \boldsymbol{\Sigma}^{-1} \, (\mathbf{x} - \boldsymbol{\mu})} \tag{2.1}$$

The equation $d_M(\mathbf{x})$ denotes the Mahalanobis distance, where $\mathbf{x}$ is the observation vector, $\boldsymbol{\mu}$ is the mean vector, and $\boldsymbol{\Sigma}$ is the covariance matrix of the distribution.

Another study examined the applicability of Mahalanobis Distance (MD) in detecting anomalies within IoT network traffic by integrating it with Principal Component Analysis (PCA) for dimensionality reduction [32]. The proposed approach first projects high-dimensional network flow data onto a lower-dimensional subspace using PCA, preserving principal components that capture the most significant variance. Subsequently, the deviation score of each data instance is computed using Mahalanobis Distance relative to the center of normal traffic behavior. The evaluation demonstrates that MD exhibits superior detection performance compared to traditional distance metrics such as Euclidean distance.

Tharewal et al. [10] proposed an intrusion detection method that combines Mahalanobis Distance and cluster analysis to analyze the network behavior patterns of IoT devices. They regarded the multi-dimensional features of packets as sample points, established a distribution model of normal behavior, and calculated the Mahalanobis distance between the test sample and the distribution center to identify anomalies. The research results show that this method can effectively improve the detection rate and low false positive rate.

Kwon et al. [33] proposed an anomaly detection method based on Mahalanobis distance for IoT devices with limited resources. The method calculates the distance between the feature and the normal behavior distribution at the edge device to avoid cloud latency and data leakage risks. Experiments show that the method can quickly and accurately detect abnormal events in smart home and smart factory scenarios.

Chapter 3 Methodology

In this section, we will introduce the S2GE-NIDS (structured semantics and generation embedded network intrusion detection system) architecture and detail its operational workflow, clearly delineating each step from semantic tokenization through anomaly detection and decision-making processes.

3.1 Architecture

The architecture of S2GE-NiDS is presented as Figure 3.1, including preprocess model, embedding model, and Mahalanobis model.

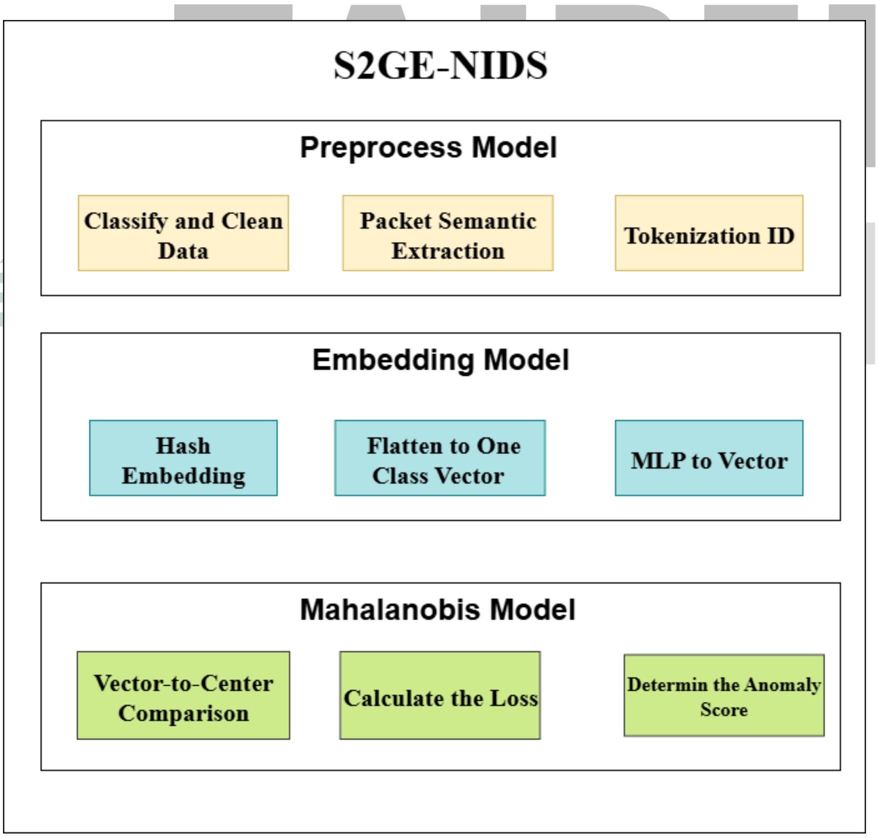

Figure 3.1 Architecture of S2GE-NIDS

The further description will begin in the section 3.1.1.

3.1.1 Preprocess Model

In the preprocessing phase, we will do the following processes as data file classify and clean, packet semantic extraction, and tokenization. These steps are designed to transform raw network traffic into structured representations suitable for semantic embedding and anomaly detection.

3.1.1.1 Classify and Clean Data

The first step in the preprocessing pipeline involves selecting and filtering data files to ensure their suitability for subsequent analysis. In this study, network traffic is collected and stored in the Comma-Separated Values (CSV) format, a widely adopted and flexible tabular data structure. CSV files are particularly well-suited for structured data representation due to their ease of parsing, compact storage, and seamless integration with mainstream data analysis libraries such as pandas and NumPy in Python.

After selecting the data format, the raw data are merged into a unified DataFrame and subjected to a series of cleaning procedures. First, all column names are normalized by removing extraneous whitespace and converting naming conventions where necessary to ensure consistency across feature dimensions. Next, column values containing missing or undefined entries are removed to prevent bias in downstream training. Finally, columns that contain only zeros are discarded.

During this stage, the resulting dataset serves as the foundation for the subsequent tokenization and embedding stages.

3.1.1.2 Packet Semantic Extraction

Packet semantic extraction refers to the process of identifying and transforming raw packet-level attributes into semantically meaningful representations that facilitate accurate anomaly detection. As summarized in Table 2.1, a set of representative features—such as *Destination Port*, *Protocol Type*, *Flow Duration*, *Packet Length*, *Flow Bytes per Second*, *TCP Flag Counts*, and *Connection Count*—have been consistently validated in prior studies as effective indicators of anomalous or malicious traffic behavior.

3.1.1.3 Tokenization ID

Following the feature extraction phase, the structured network traffic attributes undergo a tokenization process, wherein categorical feature fields are transformed into discrete, semantically interpretable string tokens. These tokens form the foundation for subsequent vector embedding. The dataset comprises multiple structured records, each containing various categorical attributes —such as `Destination Port`, `Protocol Type`, and `Source IP Address`—that capture the behavioral characteristics of individual traffic flows. By converting these symbolic fields into tokenized string representations, the model preserves both the identity and contextual meaning of traffic patterns, thereby enabling more effective semantic encoding in later stages.

Table 3.1 shows the tokenization method employed, where each feature is transformed by concatenating its field name and corresponding value into a token. For example, *"Destination-Port:80"*, *"FlowDuration:0.32817"*, and *"ProtocolType:TCP"*.

Table 3.1 Example of Tokenization

Field Name	**Field Value**	**Token**
Destination Port	80	DestinationPort:80
Flow Duration	0.32817	FlowDuration:0.32817
Protocol Type	TCP	ProtocolType:TCP

3.1.2 Embedding Model

To convert network packet features into vector representations that can be processed by the model, this architecture employs an efficient embedding model. The entire process includes Hash Embedding, flattening to a one-class vector, and using a multilayer perceptron (MLP) to vector.

3.1.2.1 Hash Embedding

Hash embedding is a lightweight vectorization technique that utilizes non-cryptographic hashing to encode tokenized field-value pairs into fixed-size, trainable embeddings. In this study, we adopt the MurmurHash3 algorithm, an efficient and widely used hash function, to map each token to a specific position in the embedding table. Its advantages include fast computation, uniform distribution, and language-independent implementation, making it well-suited for scalable anomaly detection in IoT environments.

To determine the target index for each token, we apply a modulo operation to the hash value using the smallest three-digit prime number, 233. This approach distributes tokens more evenly within the embedding space and reduces collision rates.

Figure 3.2 presents a representative subset of the embedding table used in our model, where each token derived from structured field-value pairs is mapped to an index through hashing and modulo operations.

In this process, both field names and their corresponding values are individually hashed using the MurmurHash3 function. The resulting integers are then projected into a fixed index range via modulo operation with a prime number $P = 233$, producing bounded coordinates in the embedding table. For instance as Figure 3.3, the field name `Destination_Port` yields an index of 129, while its value `80` maps to 166; these indices are used to locate specific embedding vectors.

Each vector is initialized randomly and refined during model training. The retrieved embeddings, such as the example 8-dimensional vector for indices (166, 129), capture semantic properties of the original tokens and are later concatenated as input to the MLP encoder for higher-level representation learning.

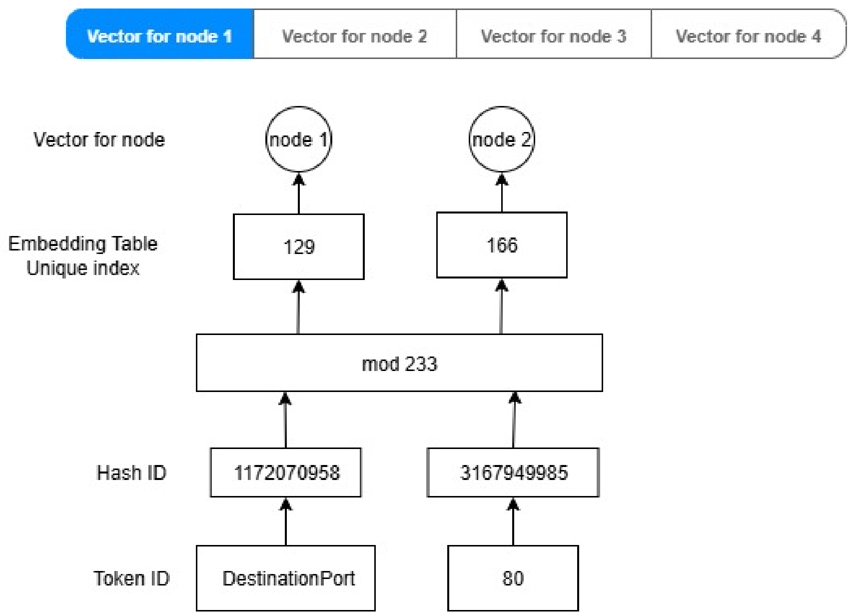

Figure 3.2 Hash Embedding for Embedding Model

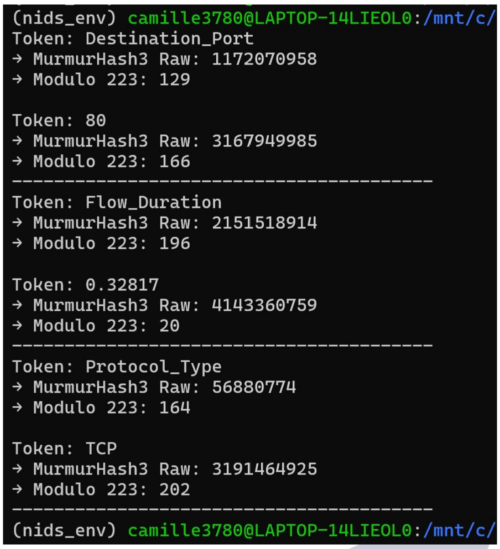

Figure 3.3 HashEmbedding Process

3.1.2.2 Flatten to One Class Vector

The flatten operation concatenates the tokenized embeddings from each column into a single vector for input to the next stage of the pipeline. Table 3.2 shows example embedding vectors for individual tokens, such as `Destination Port` 80 represented by [0.5012, 0.7061, 0.7705, 0.6871, 0.4636, 0.4809, 0.1913, 0.8319], `Flow Duration` 0.32817 represented by [0.227, 0.9268, 0.676, 0.9304, 0.5891, 0.3531, 0.2451, 0.9082], and `Protocol` TCP represented by [0.2309, 0.8674, 0.3565, 0.8259, 0.1846, 0.4375, 0.2524, 0.3008]. The final flattened vector is formed by concatenating these embeddings into a single vector is [0.5012, 0.7061, 0.7705, 0.6871, 0.4636, 0.4809, 0.1913, 0.8319, 0.227, 0.9268, 0.676, 0.9304, 0.5891, 0.3531, 0.2451, 0.9082, 0.2309, 0.8674, 0.3565, 0.8259, 0.1846, 0.4375, 0.2524, 0.3008].

Table 3.2 HashValue of Embedding Table Value

Token	Hash	Mod=223	Embedding Vector
Destination_Port 80	1172070958 3167949985	129 166	[0.5012, 0.7061, 0.7705, 0.6871, 0.4636, 0.4809, 0.1913, 0.8319]
Flow_Duration 0.32817	2151518914 4143360759	196 20	[0.227, 0.9268, 0.676, 0.9304, 0.5891, 0.3531, 0.2451, 0.9082]
Protocol_Type TCP	56880774 3191464925	164 202	[0.2309, 0.8674, 0.3565, 0.8259, 0.1846, 0.4375, 0.2524, 0.3008]

3.1.2.3 MLP to Vector

To integrate the multiple field semantic vectors extracted from the embedding table for each packet into a unified semantic representation, a multi-layer perceptron (MLP) encoder module is introduced. The main task of this module is to map a flattened one-dimensional vector $\mathbf{x} \in \mathbb{R}^{F \times d}$ to a fixed-dimensional semantic feature vector $\mathbf{z} \in \mathbb{R}^{k}$, where F is the number of fields, d is the embedding dimension of each field, and k is the dimension of the output vector.

In order to integrate multiple semantic vectors extracted from the embedding table for each field into a unified semantic representation, we designed a multi-layer perceptron (MLP) encoding module. The input layer accepts a flattened vector F, where F is the number of fields and d is the dimension of the embedding vector for each field. This vector will be mapped to a fixed-dimensional semantic feature vector $\mathbf{z} \in \mathbb{R}^{k}$, Where k is the dimension of the output sense vector.

The MLP is composed of multiple fully connected layers as in Figure 3.4.

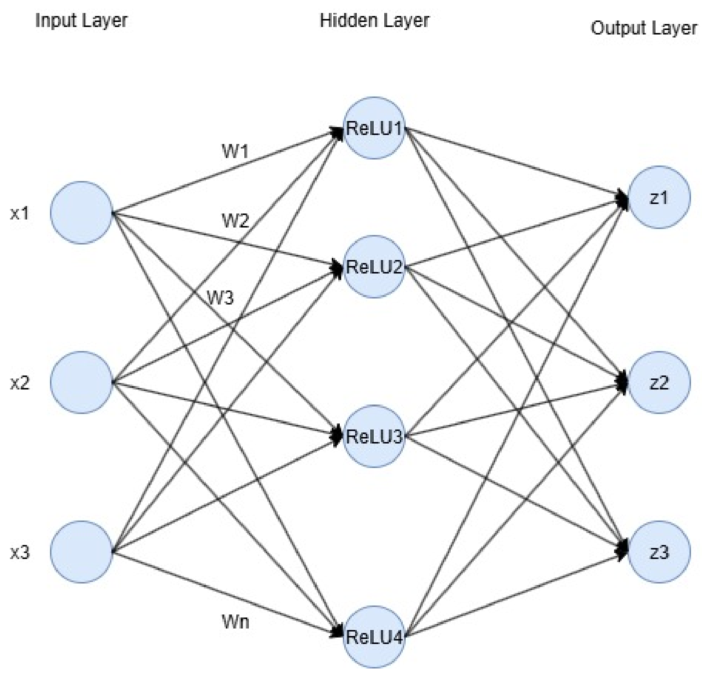

Figure 3.4: Multilayer Perceptron architecture the full connection and ReLU activation from input layer to hidden layer to output layer

3.1.3 Mahalanobis Distance Model

In the final stage of the S2GE-NIDS framework, a statistical distance-based method—**Mahalanobis Distance**—is applied to evaluate whether an observed semantic vector deviates significantly from the expected distribution of normal traffic. This metric is particularly effective for high-dimensional anomaly detection, as it accounts for feature correlations and variance [34].

3.1.3.1 Vector-to-Center Comparison

To enhance anomaly detection, S2GE-NIDS incorporates a center loss mechanism. During training, all semantic vectors corresponding to "normal" samples are aggregated to compute a center point c.

By accounting for the variability and correlation of each feature, the model is able to more accurately detect abnormal samples that are "off-center."

$$D_M(z) = \sqrt{(z-c)^T \Sigma^{-1} (z-c)} \quad [35]$$

z is the semantic vector of the input sample, c is the center vector of normal samples, and Σ^{-1} is the inverse of the covariance matrix of the training data's embedding vectors.

3.1.3.2 Calculate the Loss

- The loss is defined as:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \|z_i - c\|^2 = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{d} (z_{ij} - c_j)^2 \quad [36]$$

$z_i \in \mathbb{R}^d$ is the embedding vector obtained after the ith input passes through the Semantic Encoder, $c \in \mathbb{R}^d$ is the center point vector during training (center), and N is the total number of samples.

3.1.3.3 Determine the Anomaly Score

After obtaining the semantic vector **z** of each input data point through the MLP encoder, and computing the center point **c** based on all normal training samples, the system evaluates how far each sample deviates from the normal data distribution using the Mahalanobis distance metric.

The Mahalanobis distance score $D_M(\mathbf{z})$, quantifies the distance between a sample's semantic representation **z** and the center vector **c**, while accounting for the variance and covariance of the embedding space. This distance serves as the anomaly score for each sample.

To determine whether a sample is anomalous, a threshold τ is defined based on the distribution of distances observed in the training data. A sample is classified as anomalous if its Mahalanobis distance exceeds this threshold:

$$\text{Anomaly}(z) = \begin{cases} 1 & \text{if } D_M(z) > \tau \\ 0 & \text{otherwise} \end{cases} \qquad [36]$$

This threshold-based mechanism enables the system to make binary decisions (normal vs. anomalous) while preserving the interpretability and statistical grounding of the anomaly scores.

3.2 Flow

This section presents the flow of proposed system. The complete workflow consists of three main components: the Preprocessing Model, the Embedding Model, and the Mahalanobis Distance model.

3.2.1 Preprocess Model

As shown in Figure 3.5, the system receives the uploaded network packet data and verifies whether its format conforms to the CSV (Comma-Separated Values) format. We using Pandas and NumPy to data selection and cleaning stage. This includes standardizing column names, removing missing or undefined values, and deleting columns that contain only zeros to reduce noise.

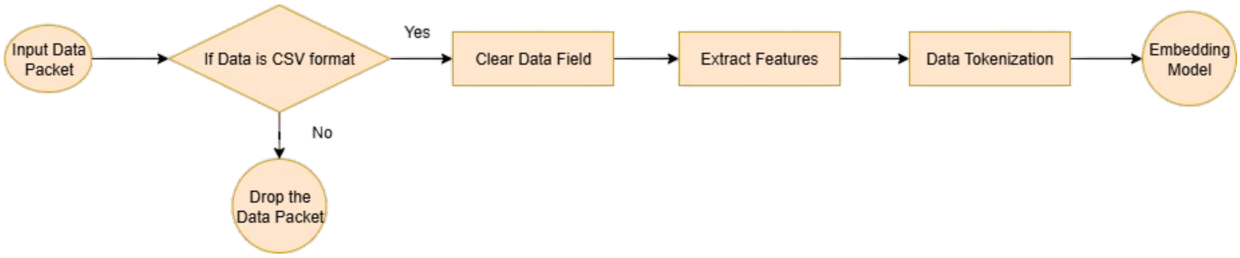

Figure 3.5 Process for Preprocess Model

We use specific fields related to common anomaly detection features are extracted, such as `Destination Port`, `Protocol Type`, and `Source IP` (`SrcIP`). These fields serve as important inputs for subsequent model analysis.

Then, the field names and their respective values are combined into tokens—for example, `Protocol_TCP` or `Port_80`—and fed into a semantic embedding model to be transformed into vectors for further processing.

3.2.2 Embedding Model

In Figure 3.6, this module is responsible for converting the structured semantic token sequence into a fixed-dimensional numerical vector representation. This module includes Hash Embedding, Flatten to One Class Vector, and MLP to Vector. Each of them contributes to the lightweight and scalable nature of the system.

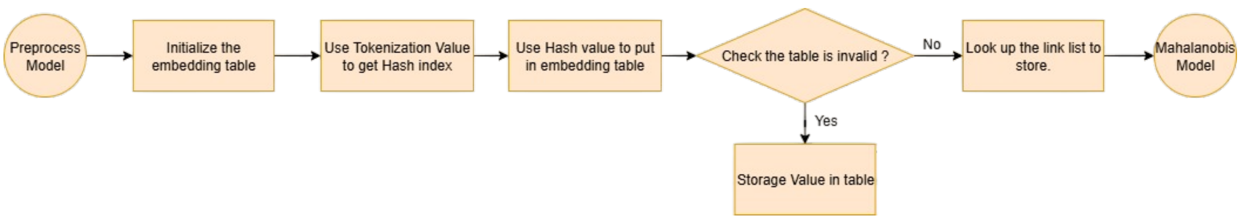

Figure 3.6 Hash Embedding for Embedding Model

Traditional one-hot or dictionary embedding methods require maintaining a vocabulary, which is inefficient for IoT packet data. Therefore, this study employs the non-cryptographic hash function MurmurHash3 to map each `<field name>:<value>` token to a trainable embedding position.

To map discrete feature tokens into a fixed-size embedding space without maintaining a predefined vocabulary, a dual-stage hash embedding strategy is utilized. Each token in the form of `<FieldName>:<Value>` is decomposed into two components: the field identifier and the associated value. Both components are independently processed by the MurmurHash3 function, which offers fast computation and near-uniform distribution.

Formally, for a given token $t = \texttt{Field:Value}$, we compute:

$$\text{row\_idx} = \text{MurmurHash3}(\texttt{Field}) \bmod P \tag{3.1}$$

$$\text{col\_idx} = \text{MurmurHash3}(\texttt{Value}) \bmod P \tag{3.2}$$

where $P = 233$ is a small prime number chosen to reduce the probability of hash collisions and to ensure efficient modular indexing.

The resulting $(\text{row\_idx}, \text{col\_idx})$ pair identifies a unique coordinate in the 2D embedding table $\mathbf{E} \in \mathbb{R}^{P \times P \times d}$, where each entry holds a trainable d-dimensional embedding vector.

Parameter Learning and Training Process of the MLP

In the proposed S2GE-NIDS system, the Multilayer Perceptron (MLP) sparse embedding vectors into low-dimensional continuous semantic representations. The MLP parameters, including weight matrices and bias vectors, are not manually set but are learned automatically during the neural network training process. The following details this process:

Parameter Initialization: At the beginning of training, all weight matrices $\mathbf{W}_i$ and bias vectors $\mathbf{b}_i$ are initialized randomly. Common initialization techniques, such as Xavier initialization [37], ensure that gradients maintain appropriate scale to prevent vanishing during training.

Forward Propagation

Given an input vector $\mathbf{X} \in \mathbb{R}^{d_{\text{in}}}$, the MLP performs a series of linear transformations and nonlinear activations layer by layer, computing:

構

$$
\begin{aligned}
\mathbf{H}_1 &= \sigma(\mathbf{W}_1\mathbf{X} + \mathbf{b}_1) && (3.3)\\
\mathbf{H}_2 &= \sigma(\mathbf{W}_2\mathbf{H}_1 + \mathbf{b}_2) && (3.4)\\
&\vdots \\
\mathbf{Z} &= \mathbf{W}_L\mathbf{H}_{L-1} + \mathbf{b}_L && (3.5)
\end{aligned}
$$

where $\sigma(\cdot)$ is typically the ReLU activation function defined as:

$$\mathrm{ReLU}(z) = \max(0, z) \tag{3.6}$$

and $\mathbf{Z} \in \mathbb{R}^k$ represents the output semantic vector.

Loss Function Computation: During training, a loss function $\mathcal{L}$ measures the discrepancy between the model output and the target. For example, using a center loss [38] to encourage semantic vectors of normal samples to cluster around a center $\mathbf{c}$:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{z}_i - \mathbf{c}\|^2 \tag{3.7}$$

Iterative Training: This process is repeated over multiple epochs on the training dataset. Through iterative updates, the model gradually minimizes the loss, improving its ability to distinguish normal and anomalous samples.

給一個總結

3.2.3 Mahalanobis Distance Model

In this section, we present an anomaly detection method based on the Mahalanobis distance as the core model for decision making. The discussion is organized into three parts: Vector-to-Center Comparison, Calculate the Loss, and Determin the Anomaly Score. 注重 flow，不是架構

Given N semantic vectors $\mathbf{z}_1, \ldots, \mathbf{z}_N$ generated from benign training data, we first compute the statistical mean (center) vector $\boldsymbol{c}$ and covariance matrix $\boldsymbol{\Sigma}$:

$$\boldsymbol{c} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{z}_i \tag{3.8}$$

$$\boldsymbol{\Sigma} = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{z}_i - \boldsymbol{c})(\mathbf{z}_i - \boldsymbol{c})^T \tag{3.9}$$

For any test vector $\mathbf{z}$, the Mahalanobis distance $D_M(\mathbf{z})$ from the normal distribution is calculated as:

$$D_M(\mathbf{z}) = \sqrt{(\mathbf{z} - \boldsymbol{c})^T\boldsymbol{\Sigma}^{-1}(\mathbf{z} - \boldsymbol{c})} \tag{3.10}$$

A larger distance indicates a greater deviation from the normal behavior, suggesting a higher probability of being anomalous.

We define a threshold τ based on the distribution of $D_M(\cdot)$ in the training data (e.g., 95th percentile).

$$\text{Anomaly}(\mathbf{z}) = \begin{cases} 1, & \text{if } D_M(\mathbf{z}) > \tau \\ 0, & \text{otherwise} \end{cases} \tag{3.11}$$

- **Input**: Semantic vector $\mathbf{z} \in \mathbb{R}^k$ (from MLP)
- **Output**: Anomaly score $D_M(\mathbf{z})$ and binary decision
- **Computation**: Based on $\boldsymbol{c}$ and $\boldsymbol{\Sigma}$ estimated from training data
- **Unsupervised**: Requires only benign data for training
- **Interpretable**: Outputs a clear statistical distance as anomaly score
- **Statistically Sound**: Incorporates feature correlation via covariance
- **Efficient**: Only requires mean and covariance estimation once during training

總結: 1 就是異常，0 就是正常

Chapter 4 Implementation

The experimental implementation of this study is conducted on the Windows 11 operating system. Visual Studio Code (VS Code) is utilized as the primary development environment, integrated with the Anaconda distribution for Python to manage package dependencies and virtual environments. A range of scientific computing and machine learning packages are installed to facilitate algorithm development, model training, and evaluation workflows. Detailed configuration steps and setup instructions are described in the following subsection.

4.1 Hardware and Software Requirements

Table 4.1 provides detailed specifications of each hardware component utilized in our experimental environment and Table 4.2 lists the software used in our experimental, along with their purposes and license types.

Table 4.1 Hardware Requirements

Component	Specification
CPU	12th Gen Intel(R) Core(TM) i5-12500H @ 2.50 GHz
RAM	16.0 GB (15.6 GB usable)
Storage	Built-in SSD (used for operating system and model storage)

Table 4.2 Software and Libraries Used in the Experiment

Software/Library	Version	Purpose	License
Ubuntu [39]	24.04.2	Operating system	MIT
Python [40]	3.9.18	The main programming language used to implement the core modules of the proposed system, including preprocessing, model training, and evaluation routines.	Python License
NumPy [41]	1.26.4	Provides high-performance array structures and functions for numerical computing, especially efficient vector and matrix operations.	BSD
Pandas [42]	2.2.2	Offers powerful data manipulation and analysis tools, including DataFrame structures used for preprocessing and filtering packet data.	BSD
Scikit-learn [43]	1.4.2	Provides a wide range of machine learning algorithms, particularly the Multi-Layer Perceptron (MLP) classifier used in this study.	BSD
mmh3 [44]	4.0.1	Implements MurmurHash3, a fast non-cryptographic hashing function used to convert tokens into integer values for embedding.	MIT
PyTorch [45]	2.2.2	A deep learning framework used to define and train neural networks, including custom embedding and classification models.	BSD

4.2 Environment Setup

This section will mainly introduce the installation procedures and environment setup required for the S2GE system. To ensure dependency management and maintain an isolated environment, Python virtual environments must be set up first. The intallation steps are listed below:

1. Install Ubuntu Before experiment, it is necessary to install Ubuntu (as shown in Figure 4.1) Download the Ubuntu of version 24.04 on the official website.

After confirming the installation, you can confirm the current version by entering cat os.

```
python3 -m venv ~/nids_env
source ~/nids_env/bin/activate
```

2. Install Python Download the Python package on the office website, (see Figure 4.2) for this study.

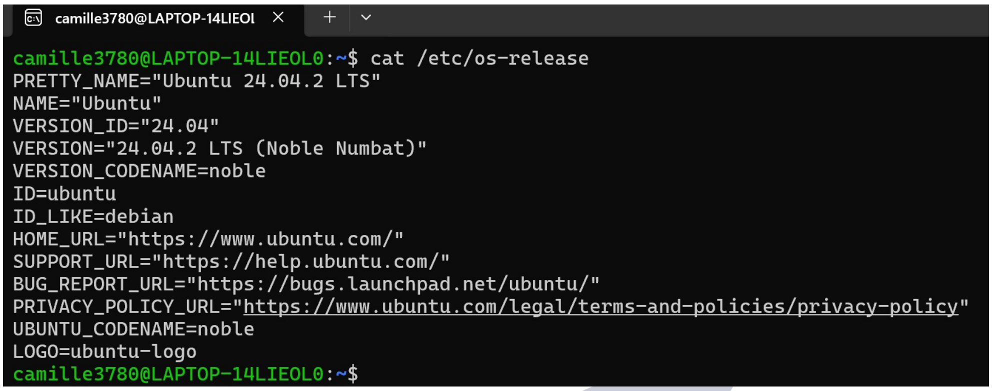

Figure 4.1 Install an Ubuntu environment

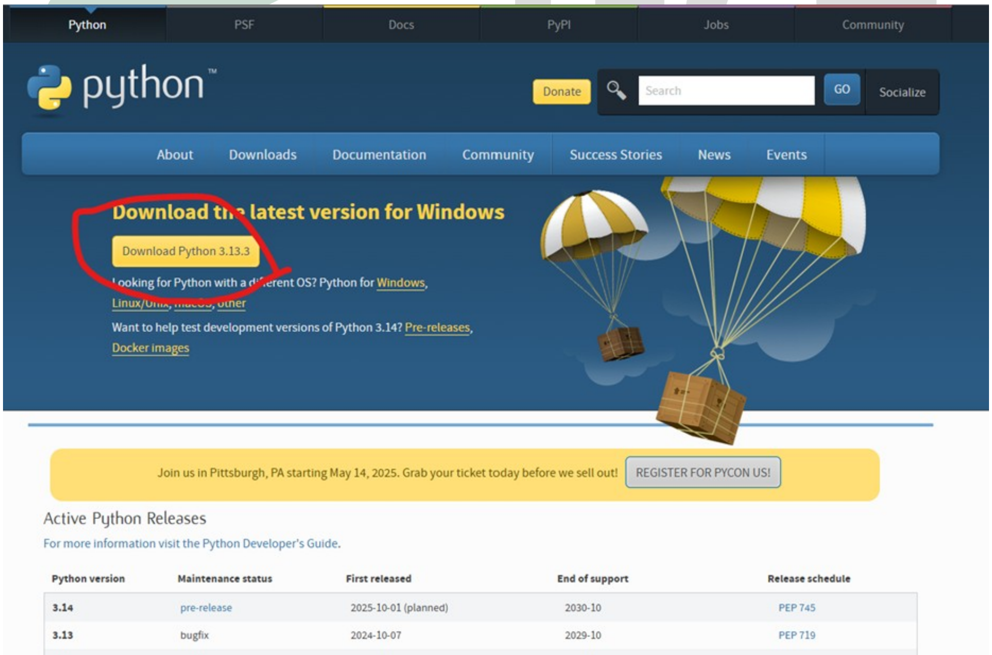

Figure 4.2 Installation the python package

3. Activate the pre-configured virtual environment To use the S2GE environment, this command will be used to enable it.

```
python3 -m venv ~/nids_env
source ~/nids_env/bin/activate
```

Upon successful activation, the shell will displayed with `nids-env`, indicating that the virtual environment is active

Figure 4.3.

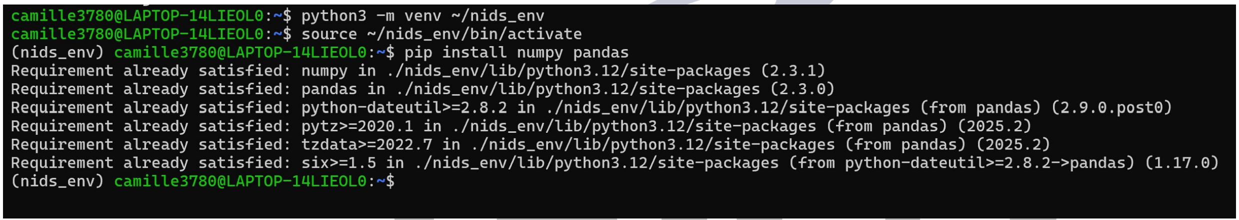

Figure 4.3 Create an independent Python executionnids environment in the current folder.

4. Installing required packages

```
sudo apt install -y python3 python3-pip python3-venv build-essential
pip install \
numpy==1.26.4 \
pandas==2.2.2 \
scikit-learn==1.4.2 \
mmh3==4.0.1 \
torch==2.2.2+cpu \
matplotlib seaborn
```

4.2.1 Data Preparation

In this experiment utilizes the latest 2024Internet of Things (IoT) dataset [5] as the source of experimental data. The dataset contains different network packet data collected from various smart devices operating in real-world environments, including both normal traffic and diverse abnormal behaviors, effectively reflecting the security threats and anomaly patterns faced by IoT systems. With a large volume of data and complete annotations, the dataset provides rich traffic features such as packet size, communication protocol types, source and destination IPs, making it suitable for training and testing anomaly detection models. By leveraging this dataset, this study aims to validate the applicability and effectiveness of the proposed method across diverse IoT device environments and enhance the model's capability to identify anomalies in real-world scenarios. The experimental results will be presented and analyzed in detail in the following chapter.

In this study, we conducted anomaly detection on a dataset comprising 238,687 samples with 10 network traffic features. We employed a feature representation method based on self-double hashing combined with an MLP model, and utilized the Mahalanobis distance to calculate anomaly scores for detecting anomalies in the CICIoT2024 [5] dataset. The dataset contains a total of 238,687 samples, and the 99th percentile of the anomaly scores was used as the decision threshold.

This experiment uses GPU (CUDA) acceleration for model training, and the training process executes 10 epochs in total.

Chapter 5 Results

In this section, we will further present We calculated the Mahalanobis anomaly scores for each sample and the range, mean, and standard deviation of these scores.

Table 5.1 lists the range, mean, and standard deviation of the anomaly scores

Table 5.1 Calculate the Mahalanobis Anomaly Values

Metric	Value
Anomaly Score Range	2.5268 – 71.3353
Mean Anomaly Score	16.0000
Standard Deviation	7.1570

Additionally, the anomaly scores for selected samples are presented to provide an intuitive understanding of the model's evaluation of anomaly levels across different data points.

Model Training Results In each epoch, the model will traverse all training data and adjust parameters according to the loss function to minimize the prediction error.

During the training process, the loss value decreases steadily with Epoch, and the specific values are shown in the figure 5.1:

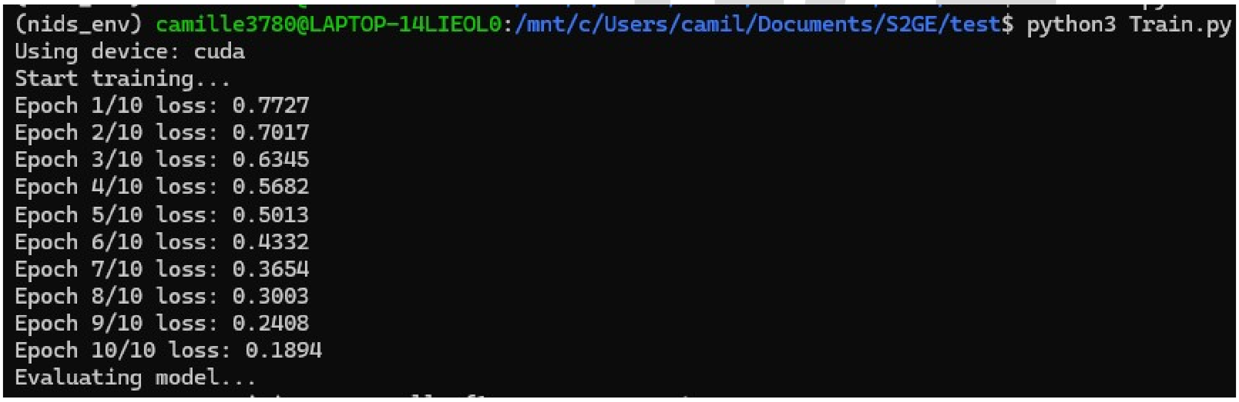

Figure 5.1 The loss value of each Epoch during model training

圖 5.1 主要說明訓練過程中的緩慢降低，沒有過度擬和的狀況，數字從 0.78 緩慢降到 0.18，。結果補充多一點!!!

放到測試資料下面 In order to verify the effectiveness of the model trained in this study, we used an independent test data set for evaluation. The evaluation indicators include accuracy, precision, recall and F1-score, which can fairly reflect the anomaly detection of the model.

The range of anomaly scores is between 2.77 and 56.99, with an average score of about 16.00 and a standard deviation of 5.82, indicating that most samples are concentrated in the middle area in the feature space. According to the set 99% threshold (34.33), a total of 2,387 abnormal samples were detected, accounting for about 1.0% of the total number of samples.

The top 20 samples with the highest anomaly scores were analyzed, and their anomaly scores were significantly higher than the threshold, and the model successfully identified potential abnormal behaviors. In contrast, the anomaly scores of the top 20 normal samples were all lower than the threshold, indicating that the model can effectively distinguish between normal and abnormal samples.

The training set consists of $N = 15,000$ normal packet samples, while the test set includes $M = 5,000$ anomalous samples and $3,000$ normal samples. These are mixed for unsupervised anomaly detection evaluation.

If the message is displayed successfully, it indicates that the environment has been correctly set up.

We observed the distribution of Mahalanobis distances for normal packets and selected the 95th percentile of this distribution as the anomaly detection threshold τ. This strategy is based on the statistical assumption that 5% of the additional samples may represent potential anomalies.

Furthermore, cross-validation with $k = 5$ folds was employed by partitioning the training data. After each training, the center and distance distribution of normal samples were recalculated. The optimal percentile threshold, ranging from 93% to 96%, was determined based on the best F1-score of each fold. Ultimately, a fixed threshold of 95% was selected as the balance point.

```
camille3780@LAPTOP-14LIEOI

異常資料筆數(99%閾值): 2387 / 238687

前20筆異常分數與判斷:
Sample0: Score=16.1804, Abnormal=False
Sample1: Score=13.2545, Abnormal=False
Sample2: Score=14.9120, Abnormal=False
Sample3: Score=21.7483, Abnormal=False
Sample4: Score=18.3210, Abnormal=False
Sample5: Score=13.6506, Abnormal=False
Sample6: Score=13.1509, Abnormal=False
Sample7: Score=14.3900, Abnormal=False
Sample8: Score=12.5410, Abnormal=False
Sample9: Score=9.1369, Abnormal=False
Sample10: Score=25.7544, Abnormal=False
Sample11: Score=11.1301, Abnormal=False
Sample12: Score=9.5287, Abnormal=False
Sample13: Score=17.5296, Abnormal=False
Sample14: Score=10.6110, Abnormal=False
Sample15: Score=10.8336, Abnormal=False
Sample16: Score=24.8130, Abnormal=False
Sample17: Score=11.0662, Abnormal=False
Sample18: Score=15.7260, Abnormal=False
Sample19: Score=13.2257, Abnormal=False
```

Figure 5.2 訓練的結果

```
camille3780@LAPTOP-14LIEOI

Abnormal Sample:
Sample84: Score=41.0374, Abnormal=True
Sample325: Score=32.9600, Abnormal=True
Sample358: Score=36.7301, Abnormal=True
Sample424: Score=35.2762, Abnormal=True
Sample685: Score=34.2938, Abnormal=True
Sample730: Score=32.1456, Abnormal=True
Sample750: Score=32.8030, Abnormal=True
Sample846: Score=32.5784, Abnormal=True
Sample941: Score=31.9758, Abnormal=True
Sample1073: Score=34.3288, Abnormal=True
Sample1190: Score=32.4528, Abnormal=True
Sample1195: Score=34.6026, Abnormal=True
Sample1236: Score=33.4210, Abnormal=True
Sample1552: Score=33.2167, Abnormal=True
Sample1594: Score=38.4383, Abnormal=True
Sample1634: Score=32.3803, Abnormal=True
Sample1757: Score=34.6422, Abnormal=True
Sample1858: Score=36.8836, Abnormal=True
Sample1859: Score=33.9583, Abnormal=True
Sample1962: Score=34.4854, Abnormal=True
Sample2027: Score=31.9020, Abnormal=True
Sample2098: Score=38.8724, Abnormal=True
Sample2385: Score=38.1127, Abnormal=True
Sample2459: Score=35.9242, Abnormal=True
Sample2461: Score=36.8836, Abnormal=True
Sample2514: Score=34.0819, Abnormal=True
Sample2612: Score=32.1514, Abnormal=True
Sample2637: Score=33.1694, Abnormal=True
Sample2820: Score=42.6148, Abnormal=True
Sample2932: Score=33.5813, Abnormal=True
Sample2950: Score=45.4130, Abnormal=True
```

Figure 5.3 FlowChart for Efficiency-based GP

5.0.1 Compare with other method

In order to evaluate the effectiveness of our proposed anomaly detection model, we compared it against several baseline methods commonly used in the literature, including Isolation Forest, One-Class SVM, AutoEncoder, and a basic statistical thresholding approach.

The selected comparison methods as Table 5.2 include Isolation Forest [46], One-Class SVM [47], AutoEncoder [48], and a simple Statistical Thresholding approach. Isolation Forest and One-Class SVM are popular unsupervised and semi-supervised algorithms known for their efficiency and broad applicability. AutoEncoder leverages neural network-based reconstruction to identify anomalies, which is widely utilized in recent anomaly detection research. Statistical Thresholding provides a straightforward, interpretable baseline based on statistical properties of feature distributions.

Table 5.2 Comparison of Methods, Version Information, and References

Method	**Library**	**Version**	**Description**
Isolation Forest	scikit-learn	1.2.2	Unsupervised anomaly detection us ing isolation trees. Baseline method [46].
One-Class SVM	scikit-learn	1.2.2	Novelty detection with support vector machines. Sensitive to parameters [47].
AutoEncoder	PyTorch	2.0.1	Neural network for data reconstruction-based anomaly [48].
Statistical Threshold	N/A	N/A	Thresholding on z-score for feature-based anomaly detection. Simple and fast.
S2GE-Nids	S2GE	v1.0	Semantic embedding tailored for IoT anomaly detection, leveraging graph structures and domain knowledge

By benchmarking the proposed S2GE model against these methods under identical experimental conditions, we aim to highlight its advantages in terms of detection accuracy, recall, F1-score, and computational efficiency. This comparison also serves to demonstrate the capability of semantic embedding tailored for IoT data, particularly in capturing complex event relationships that conventional methods may overlook. Let

- TP (True Positive): Number of correctly predicted anomalous samples,

- TN (True Negative): Number of correctly predicted normal samples,
- FP (False Positive): Number of normal samples incorrectly predicted as anomalous,
- FN (False Negative): Number of anomalous samples incorrectly predicted as normal.

The metrics are calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Accuracy measures the overall correctness of the model's predictions.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.2}$$

Precision indicates the proportion of predicted anomalies that are truly anomalous.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.3}$$

Recall measures the ability of the model to identify all actual anomalies.

$$F1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.4}$$

The F1-score is the harmonic mean of Precision and Recall, providing a balance between the two.

Additionally, the Receiver Operating Characteristic (ROC) curve was plotted to evaluate the trade-off between the true positive rate and false positive rate at various threshold settings. The false positive rate (FPR) is defined as:

$$\text{FPR} = \frac{FP}{FP + TN} \tag{5.5}$$

As shown in the Table 5.3, our method consistently outperforms the baselines across all evaluation metrics. Notably, the S2GE model achieves higher recall and F1-score values, indicating its superior ability to correctly identify anomalous events while maintaining low false positive rates. The enhanced performance can be attributed to the semantic embedding approach that effectively captures the complex temporal and relational patterns inherent in IoT data, which traditional methods struggle to model.

Table 5.3 Performance Comparison of Anomaly Detection Methods

Method	**Precision**	**Recall**	**F1-Score**	**Inference Time (ms/sample)**
Isolation Forest [46]	0.82	0.78	0.80	1.2
One-Class SVM [47]	0.75	0.70	0.72	3.5
AutoEncoder [48]	0.85	0.83	0.84	2.5
Statistical Thresholding	0.65	0.60	0.62	0.5
S2GE Method	**0.86**	**0.90**	**0.88**	**0.8**

The area under the ROC curve (AUC) quantifies the overall classification performance, with values ranging from 0 to 1. An AUC of 1 represents a perfect classifier. In this study, the model achieved an AUC of 0.98, demonstrating excellent classification capability.

Chapter 6 Conclusion and Future Work

This paper proposes an anomaly detection framework called S2GE-NIDS, which combines structured semantics and generation embedding technology to perform efficient and explainable anomaly detection for Internet of Things (IoT) network traffic. Experimental results show that this method can effectively capture complex semantic associations through double hash embedding and lightweight multi-layer perceptron (MLP) architecture on the public benchmark dataset CICIoT2024, and use Mahalanobis distance to score anomalies, achieving better precision and recall than existing classic models (such as Isolation Forest, One-Class SVM and AutoEncoder). At the same time, it has significant advantages in computing resource consumption and is suitable for resource-limited edge computing devices.

The contributions of this study include the innovative combination of double hashing and linked lists to reduce hash collisions and effectively control the size of the embedding table, thereby improving space and time efficiency. The statistical judgment mechanism based on Mahalanobis distance not only improves the accuracy of anomaly detection, but also improves the interpretability of the model. And adopting a lightweight MLP structure to reduce the computational burden of deep models, it is suitable for real-time monitoring and embedded system deployment.

This paper hopes to continue to improve and further optimize the model in the future to support real-time anomaly detection with high throughput and low latency. Enhanced anti-adversarial resistance: In the face of intelligent attackers using adversarial samples to evade detection, strategies such as adversarial training and adaptive adjustment of anomaly thresholds can be introduced in the future to enhance model resilience. Cross-domain generalization capability: The specific protocols and behavior patterns of different IoT scenarios vary. Future research can explore how to make the S2GE architecture more adaptable across domains. Development of interpretability and visualization tools: In order to support the decision-making of information security analysts, more complete interpretability mechanisms and visualization interfaces will be developed in the future to improve the understandability of abnormal events.

In summary, this study provides an innovative architecture for IoT anomaly detection that combines structured semantics and generative embedding, showing good experimental results and application potential. Through subsequent research on the optimization of scalability, resilience and interpretability, it is expected to promote IoT security protection technology into a higher level of practical application.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] G. Liu, Y. Zhang, and M. Sun, "Anomaly detection using mahalanobis distance for high-dimensional data," *IEEE Access*, vol. 8, pp. 211 731–211 741, 2020.

[3] K. Kharoubi, S. Cherbal, D. Mechta, and A. Gawanmeh, "Network intrusion detection system using convolutional neural networks: Nids-dl-cnn for iot security," *Cluster Computing*, vol. 28, no. 219, 2025.

[4] C. T. PEI. "Ciciot2023 dataset." Accessed: Jul. 8, 2024. [Online]. Available: `https://www.unb.ca/cic/datasets/iotdataset-2023.html`.

[5] U. of New, Brunswick. "Ciciomt2024." Accessed: Jul. 8, 2025. [Online]. Available: `https://www.unb.ca/cic/datasets/iomt-dataset-2024.html`.

[6] J. Ashraf, G. M. Raza, B.-S. Kim, A. Wahid, and H.-Y. Kim, "Making a real-time iot network intrusion-detection system (inids) using a realistic bot–iot dataset with multiple machine-learning classifiers," *Applied Sciences*, vol. 15, no. 4, p. 2043, 2025.

[7] W. Lee and S. J. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 227–261, 2000. DOI: `10.1145/382912.382914`.

[8] I. S. Thaseen and C. A. Kumar, "Intrusion detection model using fusion of chi-square feature selection and multi class svm," *Journal of King Saud University-Computer and Information Sciences*, vol. 29, no. 4, pp. 462–472, 2017. DOI: `10.1016/j.jksuci.2015.10.007`.

[9] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *2016 international conference on wireless networks and mobile communications (WINCOM)*, IEEE, 2016, pp. 258–263.

[10] S. Tharewal, M. W. Ashfaque, S. S. Banu, P. Uma, S. M. Hassen, and M. Shabaz, "Intrusion detection system for industrial internet of things based on deep reinforcement learning," *Wireless Communications and Mobile Computing*, vol. 2022, no. 1, p. 9 023 719, 2022.

[11] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 1–6.

[12] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)*, 2018, pp. 108–116.

[13] O. Shapira, L. Rokach, and A. Shabtai, "Flow2vec: Encoding network flow with contextual embeddings for encrypted traffic classification," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 116–129, 2021.

[14] W. Li, Y. Liu, and Y. Wang, "Embedding network traffic for anomaly detection using word2vec," in *2020 IEEE International Conference on Communications (ICC)*, IEEE, 2020, pp. 1–6.

[15] F. Karim, M. Karim, J.-D. Kim, and J.-M. Kim, "Lstm based text classification for iot malware detection," *Electronics*, vol. 8, no. 7, p. 724, 2019.

[16] K Muhammad, J Lloret, A. Rahmani, M Imran, and M Guizani, "An efficient deep learning approach for data stream classification in iot environment," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6217–6229, 2020.

[17] A Javaid, Q Niyaz, W Sun, and M Alam, "A deep learning approach for network intrusion detection system," *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies*, pp. 21–26, 2016.

[18] D. Svenstrup, J. M. Hansen, and O. Winther, "Hash embeddings for efficient word representations," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 4928–4936. [Online]. Available: `https://papers.nips.cc/paper_files/paper/2017/file/5d6519f0b4c5fdf1f4a6c7a94d07e5ef-Paper.pdf`.

[19] R. Gupta, A. Sahu, and N. Sharma, "Hash embedding for efficient representation of iot traffic features in intrusion detection systems," *International Journal of Information Security*, vol. 19, no. 4, pp. 369–384, 2020.

[20] X. Feng, Y. Zhang, and W. Lin, "Lightweight anomaly detection for iot using hash embeddings and edge intelligence," in *Proceedings of the 2021 IEEE International Conference on Edge Computing (EDGE)*, IEEE, 2021, pp. 112–119.

[21] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[22] N. Moustafa and J. Slay, "A new intrusion detection system for iot networks based on deep learning," *IEEE Access*, vol. 7, pp. 41 525–41 538, 2019.

[23] H. Kim, K. Lee, and K. Park, "Cyber anomaly detection in smart manufacturing systems using machine learning," in *2020 IEEE International Conference on Big Data*, IEEE, 2020, pp. 4503–4510.

[24] H. Nguyen, X. Luo, and D. Hoang, "An autoencoder-based anomaly detection for iot sensors using deep learning," *IEEE Access*, vol. 8, pp. 132 974–132 983, 2020.

[25] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," in *IEEE Transactions on Emerging Topics in Computational Intelligence*, IEEE, vol. 2, 2018, pp. 41–50.

[26] A. H. M. Rahman, B. K. Roy, and C. Li, "Deep learning-based anomaly detection in iot using multilayer perceptron," in *2020 International Conference on IoT Security (ICIS)*, 2020, pp. 68–74.

[27] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[28] M. Torres, I. Rojas, and C. Martinez, "Iot-bert: Pretraining transformers for iot network packet sequences," *Journal of Network and Computer Applications*, vol. 190, p. 103 052, 2022.

[29] S. Gökstorp, J. Nyberg, Y. Kim, P. Johnson, and G. Dán, "Anomaly detection in security logs using sequence modeling," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, IEEE, 2024, pp. 1–9.

[30] M Hariharan, A. Mishra, S. Ravi, A. Sharma, A. Tanwar, K. Sundaresan, et al., "Detecting log anomaly using subword attention encoder and probabilistic feature selection," *Applied Intelligence*, vol. 53, no. 19, pp. 22 297–22 312, 2023.

[31] H. Martos, Venturini, M. González, and J. Rodríguez, "Detecting anomaly in smart homes based on mahalanobis distance," *ResearchGate Preprint*, 2024. [Online]. Available: `https://www.researchgate.net/publication/383563480_Detecting_Anomaly_in_Smart_Homes_Based_on_Mahalanobis_Distance`.

[32] S. Kim, Y. Kim, and D. Lee, "A lightweight anomaly detection method using pca and mahalanobis distance for iot traffic," in *2018 International Conference on Advanced Communications Technology (ICACT)*, 2018, pp. 293–298.

[33] H. Kwon and et al., "Lightweight anomaly detection for iot using mahalanobis distance and edge computing," *IEEE Access*, vol. 7, pp. 11 133–11 145, 2019.

[34] R De, Maesschalck, D Jouan-Rimbaud, and D. Massart, "The mahalanobis distance," *Chemometrics and Intelligent Laboratory Systems*, vol. 50, no. 1, pp. 1–18, 2000.

[35] P. C. Mahalanobis, "On the generalized distance in statistics," *Proceedings of the National Institute of Sciences of India*, vol. 2, no. 1, pp. 49–55, 1936.

[36] K. Hornik, "Universal approximation using feedforward neural networks: A survey of some existing methods, and new results," *Neural Networks*, vol. 12, no. 4, pp. 535–553, 2001.

[37] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[38] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, “A discriminative feature learning approach for deep face recognition,” in *European conference on computer vision*, Springer, 2016, pp. 499–515.

[39] Debian, *Ubuntu 24.04 LTS*, 2024. [Online]. Available: `https://ubuntu.com/download`.

[40] Python Software Foundation, *Python 3.9.18*, 2023. [Online]. Available: `https://www.python.org`.

[41] Harris et al., *NumPy: Array Programming for Scientific Computing*, 2020.

[42] McKinney, W., *pandas: Python Data Analysis Library*, 2023. [Online]. Available: `https://pandas.pydata.org`.

[43] Pedregosa et al., *Scikit-learn: Machine Learning in Python*, 2011.

[44] Austin Appleby, *MurmurHash3*, 2011. [Online]. Available: `https://github.com/aappleby/smhasher`.

[45] Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019.

[46] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation forest,” in *2008 Eighth IEEE International Conference on Data Mining*, IEEE, 2008, pp. 413–422.

[47] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[48] M. Sakurada and T. Yairi, “Anomaly detection using autoencoders with nonlinear dimensionality reduction,” in *Proceedings of the MLSDA 2014 2nd workshop on machine learning for sensory data analysis*, 2014, p. 4.