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 and Transformer-based models [2] 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 **S2GE-NIDS** (Structured Semantics and Generation Embedded Network Intrusion Detection System) 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 [3]. 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 [4].

The proposed S2GE-NIDS framework offers several advantages over conventional intrusion detection systems. First, it uses a hash-based embedding approach; our common hash tables are usually used for efficient data search and storage, and the combination of double hashing and linked list serial nodes can compress and pre-allocate to save index space and reduce dynamic memory configuration costs [3], [5]. 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 [4], [6]. 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 [7], 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 (Structured Semantics and Generation Embedded Network Intrusion Detection System). 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 language tags.

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. [8] proposed NIDS-DL-CNN, 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 [9] and CICIoMT2024 [10]. 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. [11] introduced a real-time 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. [12] 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. [13] proposed a method combining feature selection with multi-class 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 [14]	Specific port targets (e.g., 22, 23, 80, 443) are often associated with attacks. Abnormal access to these ports may suggest behaviors such as scanning, DDoS, or brute-force intrusion.
Protocol Type [15]	Sudden increases in uncommon protocols (e.g., ICMP, UDP) may reveal attempts to exploit protocol vulnerabilities or bypass filters.
Duration / flow_duration [15]	Unusually short or excessively long session durations may be indicative of attacks such as probing or data exfiltration.
Packet Length [14]	Anomalies in packet size—whether fixed, too long, or too short—often reflect malicious traffic like botnet propagation or worms.
Source IP / Destination IP [16]	Repeated access from abnormal IP addresses, or sudden surges in novel IP sources, are indicative of scanning, spoofing, or DDoS activity.
Flow Bytes per Second [14]	Sharp fluctuations surges or drops in flow byte rate may suggest DoS attacks or unauthorized data transfer.
TCP Flags [17]	Unusual combinations (e.g., SYN, FIN, RST) can indicate stealth scans or TCP-based flooding.
Number of Connections [14]	A large number of new connections established by a single IP in a short time often reflects worm propagation or botnet coordination.

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. [18] 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. Similarly et al. [19] 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. [20] 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. [21] 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. [22] 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	DstPort80
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 [23], 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. The main method is to combine the (field name, field value) of each data sample and pass it through a hash function such as MurmurHash3 to obtain a set of row/col indexes. This set of indices is then used to query a multi-dimensional embedding table, where an initial random, trainable semantic vector is stored at each position [3].

As Gupta et al. [24] 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. [25] 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 [26].

In recent years, MLP-based anomaly detection methods have been employed in various domains, including network security [27], industrial control systems [28], and IoT environments [29]. 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. [27] 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. [29] 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. [30] 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. [31] 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. [22] 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 [32].

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 [19]. 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. [18] proposed Flow2Vec, which encodes sequences of network events into dense vectors, improving anomaly detection in encrypted traffic.

Torres et al. [33] 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$_{I}oTdataset.$

Rahman et al. [34] 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. [35] 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. [36] 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 [37]. 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. [15] 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. [38] 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.

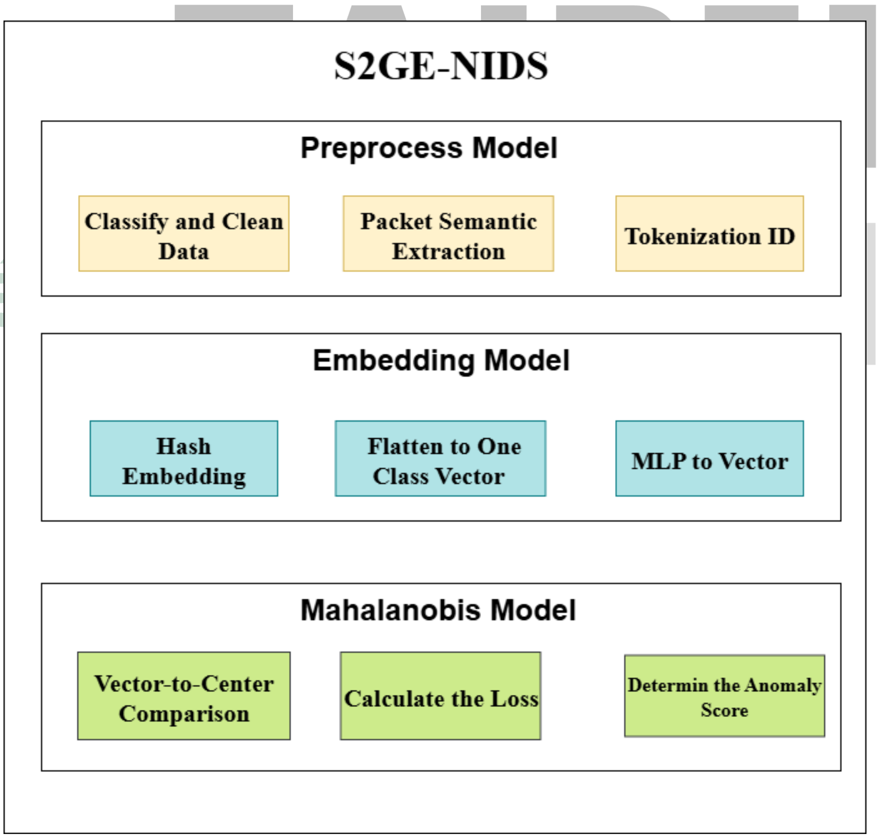


Figure 3.1 Architecture of S2GE-NIDS

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 obtain the final vector representation.

3.1.2.1 Hash Embedding

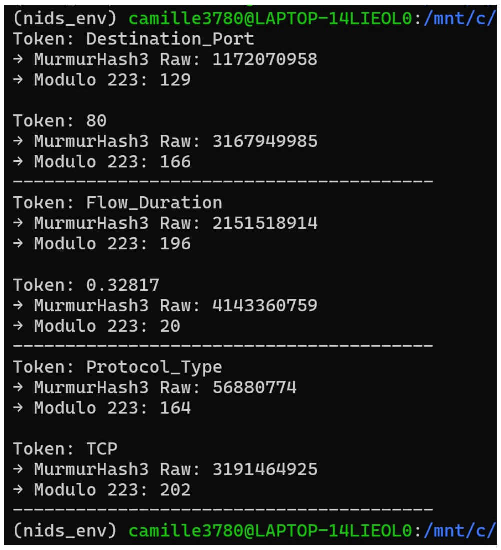
Hash embedding is a lightweight vectorization technique that utilizes non-cryptographic hashing to encode tokenized field-value pairs into fixed-size, trainable embeddings. In this study, we adopt the MurmurHash3 algorithm, an efficient and widely used hash function, to map each token to a specific position 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, 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.



```
(nids_env) camille3780@LAPTOP-14LIEOL0:/mnt/c/
Token: Destination_Port
→ MurmurHash3 Raw: 1172070958
→ Modulo 223: 129

Token: 80
→ MurmurHash3 Raw: 3167949985
→ Modulo 223: 166
----------------------------------------
Token: Flow_Duration
→ MurmurHash3 Raw: 2151518914
→ Modulo 223: 196

Token: 0.32817
→ MurmurHash3 Raw: 4143360759
→ Modulo 223: 20
----------------------------------------
Token: Protocol_Type
→ MurmurHash3 Raw: 56880774
→ Modulo 223: 164

Token: TCP
→ MurmurHash3 Raw: 3191464925
→ Modulo 223: 202
----------------------------------------
(nids_env) camille3780@LAPTOP-14LIEOL0:/mnt/c/
```

Figure 3.2 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 $[0.5012, 0.7061, 0.7705, 0.6871, 0.4636, 0.480$ `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: [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

|l|r|r|X|

Token	Hash	Mod=223	Embedding Vector (16-d)
Destination_Port	1172070958	129	[0.5012, 0.7061, 0.7705, 0.6871, 0.4636, 0.4809, 0.1913, 0.8319]
80	3167949985	166	
Flow_Duration	2151518914	196	[0.227, 0.9268, 0.676, 0.9304, 0.5891, 0.3531, 0.2451, 0.9082]
0.32817	4143360759	20	
Protocol_Type	56880774	164	[0.2309, 0.8674, 0.3565, 0.8259, 0.1846, 0.4375, 0.2524, 0.3008]
TCP	3191464925	202	

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 $\mathbf{x} \in R^{F\times d}$ to a fixed-dimensional semantic feature vector $\mathbf{z} \in R^{k}$, where F is the number of fields, d is the embedding dimension of each field, and k is the dimension of the output vector.

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 [6].

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.”

$$D_M(z) = \sqrt{(z-c)^T \Sigma^{-1} (z-c)} \tag{39}$$

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 Vector-to-Center Comparison

- The loss is defined as:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} \|z_i - c\|^2 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{d}(z_{ij} - c_j)^2 \tag{40}$$

$z_i \in R^d$ is the embedding vector obtained after the ith input passes through the Semantic Encoder, $c \in R^d$ 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 $D_M(\mathbf{z})$, as defined in Equation **??**, 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:

$$\text{Anomaly}(z) = \begin{cases} 1 & \text{if } D_M(z) > \tau \\ 0 & \text{otherwise} \end{cases} \qquad [40]$$

Here, τ can be determined in several ways, such as:

- Using the mean plus k standard deviations from the training distribution (e.g., $\tau = \mu + k\sigma$).
- Setting τ based on a desired false-positive rate (e.g., the 95th percentile of $D_M(z)$ on normal samples).

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 proposed system, detailing its operational principles and experimental procedures. The complete workflow consists of three main components: the Preprocessing Model, the Embedding Model, and the Mahalanobis Distance module.

3.2.1 Preprocess Model

As shown in Figure 3.3, 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.

Subsequently, 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.

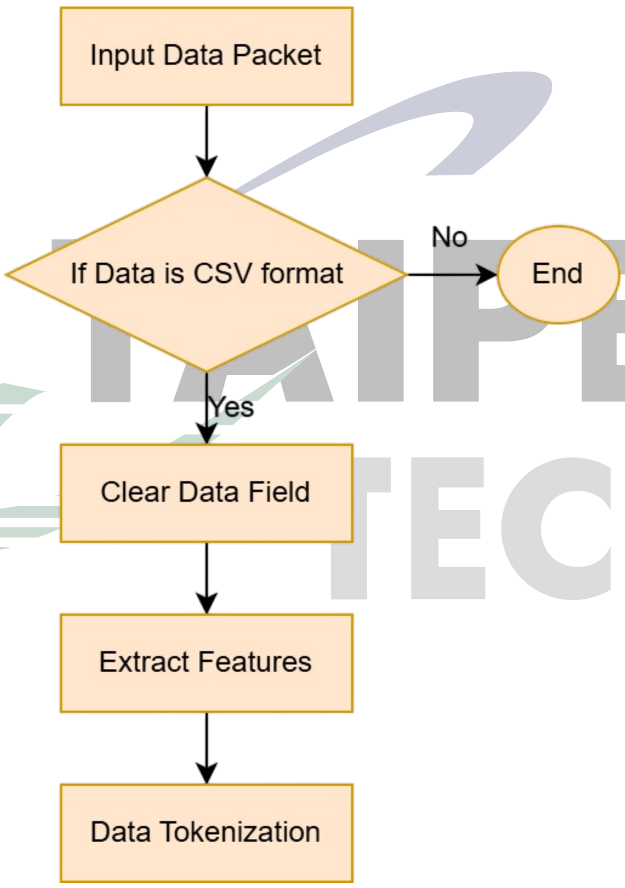


Figure 3.3 Process for Preprocess Model

Finally, 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 3.4, 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.

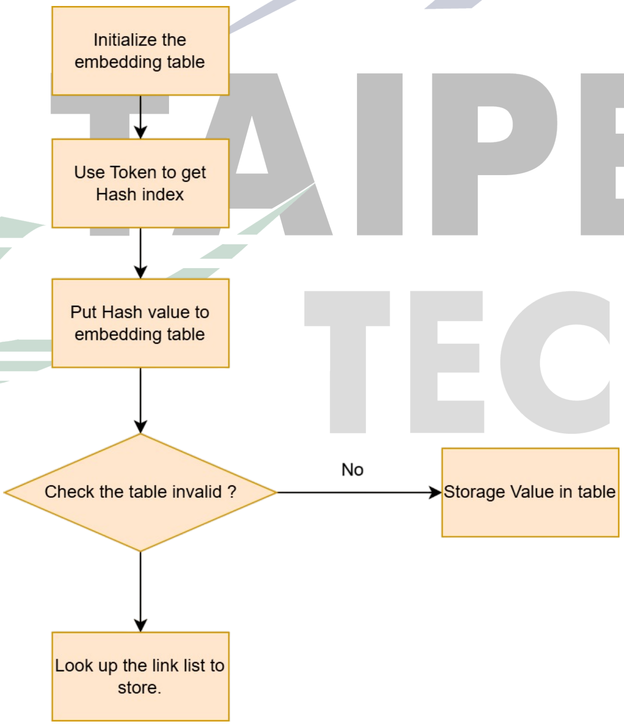

Initialize the embedding table
Use Token to get Hash index
Put Hash value to embedding table
Check the table invalid ?
No
Storage Value in table
Look up the link list to store.

Figure 3.4 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 $\mathbf{E} \in R^{P\times P\times d}$, where each entry holds a trainable d-dimensional embedding vector.

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, Vector-to-Center Distance Calculation, and Anomaly Score Determination.

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 $\boldsymbol{\Sigma}$:

$$\boldsymbol{c} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{z}_i \tag{3.3}$$

$$\boldsymbol{\Sigma} = \frac{1}{N-1}\sum_{i=1}^{N} (\mathbf{z}_i - \boldsymbol{c})(\mathbf{z}_i - \boldsymbol{c})^T \tag{3.4}$$

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

$$D_M(\mathbf{z}) = \sqrt{(\mathbf{z} - \boldsymbol{c})^T \boldsymbol{\Sigma}^{-1} (\mathbf{z} - \boldsymbol{c})} \tag{3.5}$$

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). A test vector **z** is labeled as anomalous if its distance exceeds τ:

$$\text{Anomaly}(\mathbf{z}) = \begin{cases} 1, & \text{if } D_M(\mathbf{z}) > \tau \\ 0, & \text{otherwise} \end{cases} \tag{3.6}$$

- **Input**: Semantic vector $\mathbf{z} \in R^k$ (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

Chapter 4 Implementation

The experimental implementation of this study was conducted on the Windows 11 operating system. Visual Studio Code (VS Code) was 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 were 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 Requirements

Table 4.1 provides detailed specifications and purposes of each hardware component utilized in our experimental environment.

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)

4.1.1 Software Requirements

This chapter mainly introduces the software installation process, which is divided into several steps. Table 4.2 lists the software used in our experimental setup, along with their purposes and license types.

Step 1: Installing Anaconda

Anaconda is an open-source Python platform designed for data science and machine learning development. It integrates commonly used data analysis libraries such as NumPy (numerical operations), Pandas (data processing), and Seaborn (data visualization).

Install Anaconda The installation began by double-clicking the downloaded Anaconda installer, followed by clicking “Next” to proceed to the next step (see Figure **??**).

Table 4.2 Software and Libraries Used in the Experiment

Software/Library	Version	Purpose	License
Visual Studio Code [41]	1.89.1	A lightweight and extensible code editor used as the primary integrated development environment (IDE) for editing Python scripts and managing project structure.	MIT
Anaconda Prompt [42]	2024.02	A command-line interface provided by the Anaconda distribution, used for managing Python virtual environments and installing dependencies via Conda or pip.	BSD
Python [43]	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 [44]	1.26.4	Provides high-performance array structures and functions for numerical computing, especially efficient vector and matrix operations.	BSD
Pandas [45]	2.2.2	Offers powerful data manipulation and analysis tools, including DataFrame structures used for preprocessing and filtering packet data.	BSD
Scikit-learn [46]	1.4.2	Provides a wide range of machine learning algorithms, particularly the Multi-Layer Perceptron (MLP) classifier used in this study.	BSD
mmh3 [47]	4.0.1	Implements MurmurHash3, a fast non-cryptographic hashing function used to convert tokens into integer values for embedding.	MIT
PyTorch [48]	2.2.2+cpu	A deep learning framework used to define and train neural networks, including custom embedding and classification models.	BSD

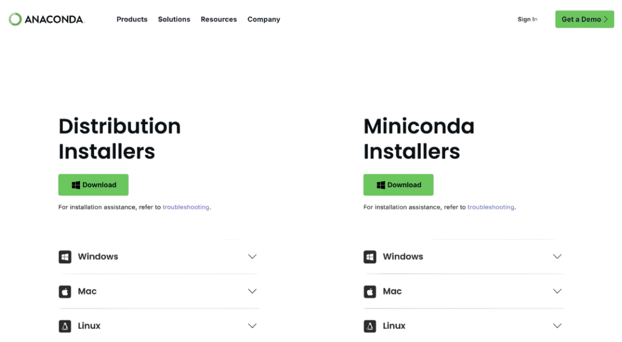
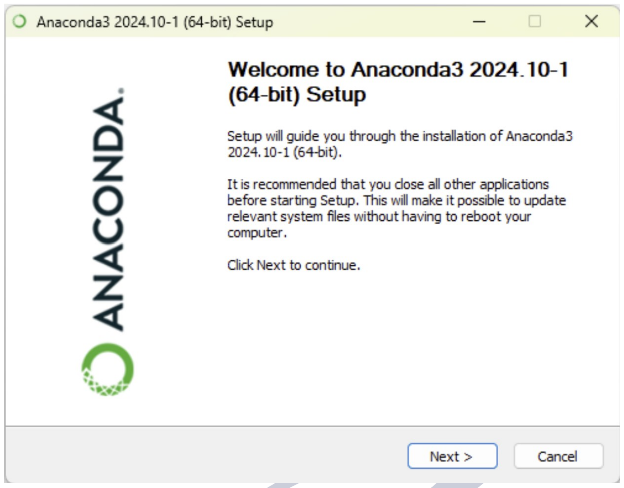


Figure 4.1 Download on Official Anaconda Website

When selecting the installation type, “Just Me” was chosen for personal use, then “Next” was clicked to continue.



Welcome to Anaconda3 2024.10-1 (64-bit) Setup

Setup will guide you through the installation of Anaconda3 2024.10-1 (64-bit).

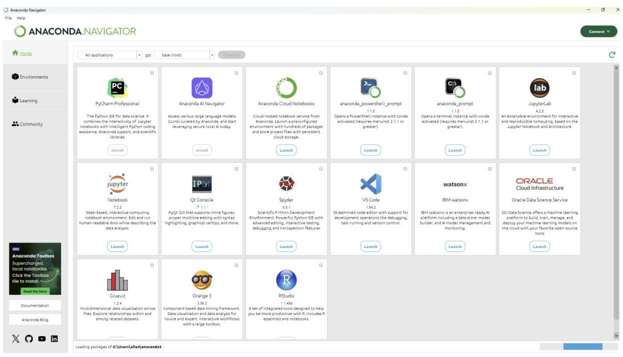
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 4.2 Installation for Anaconda

In the installation options, it is recommended not to check the option to add Anaconda to the PATH environment variable (unless specifically required). Simply click “Install” to begin the installation process.

After the installation completes, Anaconda Navigator can be found and launched from the Windows Start menu (see Figure **??**).



PyCharm Professional — The Python IDE for data science. It combines the interactivity of Jupyter notebooks with intelligent Python coding assistance, Anaconda support, and scientific libraries.

Anaconda AI Navigator — Access various large language models (LLMs) curated by Anaconda, and start leveraging secure local AI today.

Anaconda Cloud Notebooks — Cloud-hosted notebook service from Anaconda. Launch a preconfigured environment with hundreds of packages and store project files with persistent cloud storage.

anaconda_powershell_prompt 1.1.0 — Opens a PowerShell instance with conda activated (requires menuinst 2.1.1 or greater).

anaconda_prompt 1.1.0 — Opens a terminal instance with conda activated (requires menuinst 2.1.1 or greater).

JupyterLab 4.2.5 — An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook 7.2.2 — Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Qt Console 5.5.1 — PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Spyder 5.5.1 — Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

VS Code 1.94.2 — Streamlined code editor with support for development operations like debugging, task running and version control.

IBM watsonx — IBM watsonx is an enterprise-ready AI platform including a data store, model builder, and AI model management and monitoring.

Oracle Data Science Service — OCI Data Science offers a machine learning platform to build, train, manage, and deploy your machine learning models on the cloud with your favorite open-source tools

Glueviz 1.2.4 — Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3 3.36.2 — Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

RStudio 1.1.456 — A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Loading packages of **C:\Users\allen\anaconda3**...

Figure 4.3 FlowChart for Preprocess Model

Step 2: Installing Visual Studio Code

Visual Studio Code (VS Code) (see Figure **??**) is a lightweight and extensible source code editor that, when used with the Python extension, provides enhanced development capabilities. The installation package can be downloaded from the official website.

Step 3: Creating a Python Virtual Environment

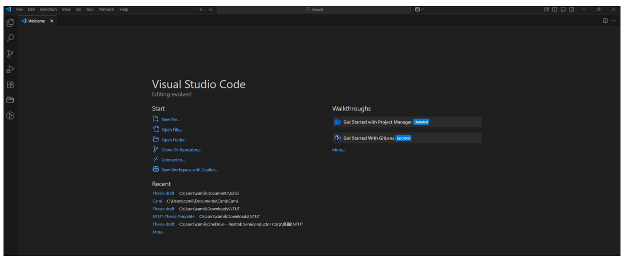

File Edit Selection View Go Run Terminal Help
Search
Welcome
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
New Workspace with Copilot...
Recent
Thesis-draft C:\Users\camil\Documents\S2GE
Cami C:\Users\camil\Documents\Cami\Cami
Thesis-draft C:\Users\camil\Downloads\NTUT
NTUT-Thesis-Template C:\Users\camil\Downloads\NTUT
Thesis-draft C:\Users\camil\OneDrive - Realtek Semiconductor Corp\桌面\NTUT
More...
Walkthroughs
Get Started with Project Manager Updated
Get Started With GitLens Updated
More...

Figure 4.4 Visual Studio Code User Interface

Use the Anaconda Prompt to create a virtual environment with the designated Python version:

```
conda create -n nids_env python=3.9

conda activate nids_env
```

Step 4: Installing Required Packages

The packages required in this study are listed below and can be installed using pip:

```
pip install numpy pandas scikit-learn matplotlib seaborn torch mmh3
```

A brief description of each package is provided in Table 4.2.

Step 5: Selecting the VS Code Interpreter

In Visual Studio Code, press `Ctrl+Shift+P` to open the command palette, then select *”Python: Select Interpreter”*. Choose the previously created `nids_env` virtual environment from the list of available interpreters.

4.1.2 Verifying the Installation

To verify the installation, create a file named `main.py` and include the following test code:

```
import numpy as np

import pandas as pd

import torch

import mmh3

print("All packages loaded successfully!")
```

Execute the script in the terminal with the following command:

```
python main.py
```

4.1.3 Dataset Description

This study utilizes the publicly available **CICIoT2023** dataset for experiments. The dataset contains normal and anomalous traffic packets from various IoT network devices, captured using Wireshark and converted into `.csv` format. In this study, three fields—**Destination Port**, **Protocol Type**, and **Source IP**—are selected as the primary input features for tokenization and embedding.

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.

Chapter 5 Results

Table 5.1 Anomaly Detection Performance Comparison

Model	**Precision**	**Recall**	**F1-Score**	**Time (ms/sample)**
Isolation Forest	0.82	0.78	0.80	1.2
AutoEncoder	0.85	0.83	0.84	2.5
S2GE-NIDS	**0.86**	**0.90**	**0.88**	**0.8**

Chapter 6 Conclusion and Future Work

References

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

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, et al., "Attention is all you need," in *Advances in neural information processing systems*, vol. 30.

[3] A. Appleby, Murmurhash3, (2011), `https://github.com/aappleby/smhasher`.

[4] 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,

[5] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pp. 1113–1120.

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

[7] K. Hornik, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366,

[8] 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,

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

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

[11] 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,

[12] 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, DOI: `10.1145/382912.382914`.

[13] 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, DOI: `10.1016/j.jksuci.2015.10.007`.

[14] 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, pp. 258–263.

[15] 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,

[16] 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*, pp. 1–6.

[17] 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)*, pp. 108–116.

[18] 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,

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

[20] 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,

[21] 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,

[22] 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,

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

[24] 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,

[25] 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, pp. 112–119.

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

[27] 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,

[28] 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, pp. 4503–4510.

[29] 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,

[30] 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, pp. 41–50.

[31] 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)*, pp. 68–74.

[32] 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,

[33] 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,

[34] 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, pp. 1–9.

[35] 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,

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

[37] 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)*, pp. 293–298.

[38] 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,

[39] 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,

[40] 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,

[41] Microsoft, Visual Studio Code, (2023). [Online]. Available: `https://code.visualstudio.com`.

[42] Anaconda, Inc., Anaconda Prompt, (2023). [Online]. Available: `https://www.anaconda.com`.

[43] Python Software Foundation, Python 3.9.18, (2023). [Online]. Available: `https://www.python.org`.

[44] Harris et al., NumPy: Array Programming for Scientific Computing, (2020).

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

[46] Pedregosa et al., Scikit-learn: Machine Learning in Python, (2011).

[47] Austin Appleby, MurmurHash3, (2011). [Online]. Available: `https://github.com/aappleby/smhasher`.

[48] Paszke et al., PyTorch: An Imperative Style, High-Performance Deep Learning Library, (2019).