TAIPEI TECH 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.

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 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 key advantages over conventional intrusion detection systems. First, it eliminates the need for manual feature engineering and vocabulary maintenance by using a hash-based embedding approach, where field-value pairs are directly encoded into semantic vectors without relying on predefined lookup tables. This design greatly simplifies the preprocessing pipeline and enhances scalability. 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, making it well-suited for deployment in real-time or resource-constrained environments such as edge devices in IoT networks. Lastly, its generalized tokenization strategy allows for wide applicability across diverse packet structures, further improving its adaptability and robustness in various network scenarios.

The structure of this paper is as follows: Chapter 2 is the relevant background knowledge about S2GE-NIDS (Structured Semantics and Generation Embedded Network Intrusion Detection System). Chapter 3 introduces the architecture and methodology of the proposed S2GE-NIDS framework, presenting each module and its rationale in detail. Chapter 4 presents the experimental setup, evaluation metrics, and results on two benchmark datasets, as well as interpretability demonstrations. Chapter 5 summarizes the main findings, limitations, and directions for future research.

Chapter 2 Related Work

This section will introduce the relevant basic knowledge, including 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. Kharoubi et al. [5] 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 and CICIoMT2024. 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 real-time intrusion detection system (NIDS) based on traditional machine learning classifiers applied to the BoT-IoT dataset. The study compared seven algorithms, including Random Forest, Artificial Neural Networks (ANN) et al., and Support Vector Machines. 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. Elrawy et al. [7] conducted a comprehensive survey of intrusion detection methodologies in IoT-based smart environments, categorizing techniques according to architectural design (centralized vs. distributed), detection strategy (signature-based, anomaly-based, or hybrid), and system layer (perception, network, application). While the survey provided valuable insights and synthesized a broad range of IDS approaches, it lacked implementation evidence and empirical comparisons, limiting its utility for practical system design. Collectively, these studies highlight the trade-offs between detection performance, computational cost, and deployment feasibility. Deep learning models offer strong accuracy but demand computational resources, while traditional classifiers provide efficiency but often lack flexibility.

As our table 2.1 summarize the eight characteristics commonly used by academia and industry are summarized below.

First, the ”target port” is one of the most direct attack indicators. IoT devices usually open a few ports for normal communication. If a large number of connections are initiated to common ports such as 22 (SSH), 23 (Telnet), 80 (HTTP), 443 (HTTPS) in a short period of time, it is very likely a precursor to malicious port scanning, brute force cracking or service denial of service attacks [8].

Next, Packet Size and Flow Bytes per Second reflect the abnormal pattern of data flow. Oversized packets may contain malicious payloads, while undersized packets may be detection messages for reconnaissance; and drastically fluctuating packet numbers or byte frequencies are common in flood attacks or covert data leakage behaviors [9].

Number of Connections is used to observe source diversity. A large number of IP connections with different destinations initiated from a single source in a very short period of time often means that the scanning tool is probing the network segment; on the contrary, if a large number of new and never-before-seen IPs appear, it may be caused by a distributed denial of service (DDoS) attack or IP spoofing [8].

Flow Duration is also critical. Scanning tools often use very short connections to detect various services; and if there is a long continuous connection, it means that the attacker is performing a long-term data exfiltration or maintaining communication with the control server (C&C) to instigate deeper penetration [9].

Protocol Type and TCP Flags help to further interpret the behavior pattern. IoT devices generally operate mainly based on TCP/UDP. If a large number of ICMP (such as ping) or other rare protocols appear, it means that the attacker may be exploiting protocol weaknesses or trying to circumvent security protection. In addition, abnormal flag combinations such as the simultaneous appearance of SYN and FIN, or a large number of RST packets in a short period of time, are typical features of stealth scanning or specific TCP attacks [9].

Finally, ”Packet Length” can reflect more subtle anomalies: when a device repeatedly sends packets of a fixed length that are too long or too short, it is very likely that the attacker has implanted malicious data in it, or is performing a forgery and spreading attack.

Combining the above eight features, the security system can detect and intercept various IoT attack behaviors at the earliest stage, providing a solid basis for the construction of subsequent intrusion detection models.

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

Feature	**Description**
Destination Port[10]	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.
Flow Duration[11]	Extremely short or long connection durations within brief timeframes may signal scanning activity or data exfiltration.
Total Forward Packets[12]	Unusually high or low packet counts in one direction may indicate abnormal sessions or flooding behavior.
Packet Length[10]	Anomalies in packet size—whether fixed, too long, or too short—often reflect malicious traffic like botnet propagation or worms.
Protocol Type[11]	Sudden increases in uncommon protocols (e.g., ICMP, UDP) may reveal attempts to exploit protocol vulnerabilities or bypass filters.
Source IP / Destination IP[13]	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[10]	Sharp fluctuations—surges or drops—in flow byte rate may suggest DoS attacks or unauthorized data transfer.
TCP Flags[12]	Unusual combinations (e.g., SYN, FIN, RST) can indicate stealth scans or TCP-based flooding.
Number of Connections[10]	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

Recent advances in IoT network security have explored various lightweight feature representations to enable efficient anomaly detection on resource-constrained devices. One promising direction is the use of token-based features, where sequences from network traffic—such as packet headers, payloads, or session patterns—are tokenized and analyzed using statistical or learning-based models.

In 2018, Yair Meidan et.al introduced in IEEE Pervasive Computing [14]. Their method segmented the raw traffic of different IoT devices into fixed-length feature vectors, which were then tokenized based on frequency and count. A deep autoencoder was trained to learn latent representations of normal behavior. On various real-world devices such as baby monitors and thermostats, their approach achieved a detection accuracy of 99.8% with a false alarm rate below 0.5%.

In 2020, Chen Li et al. proposed a sequence modeling approach for IoT anomaly detection in their work []. They treated URL paths and DNS queries as textual sequences, applied n-gram segmentation and TF-IDF vectorization, and trained traditional classifiers such as Random Forest and Support Vector Machine (SVM). Their method achieved F1-scores exceeding 96% on malicious domain detection tasks, demonstrating the strong discriminative power of token frequency patterns.

More recently, in 2022, Miguel Torres, Isabel Rojas, and Carlos Martinez introduced [15]. Their model utilized a Transformer-based architecture trained via self-supervised learning on tokenized packet headers and payloads. IoT-BERT learned context-aware embeddings of network behavior and achieved over 98% accuracy on the TON_IoT dataset for tasks such as anomaly classification and attack attribution.

These studies collectively highlight the effectiveness of token-based encoding strategies in modeling IoT traffic, offering a promising foundation for scalable, efficient, and accurate anomaly detection.

2.3 Hash Embedding

Svenstrup et al. [16] proposed an efficient method combining classic embedding and hashing trick at NeurIPS. Because traditional embedding needs to create independent parameter vectors for each token in the vocabulary, when the vocabulary is very large, the number of parameters and memory requirements increase sharply. The hashing trick maps high-dimensional features to a fixed-size table through a single hash function, but collisions are difficult to control and may affect the results. The principle is to use multiple hash functions to map each token to different vectors in a predefined large vector space. This method allows parameter sharing and can adjust the mapping weights according to the data, reducing the negative impact of collision, and the memory consumption is much less than the full embedding.

Gupta et al. [17] proposed a hash embedding-based representation method for protocol-level IoT traffic, focusing on categorical fields like destination ports, device types, and payload signatures. Their system integrates a multi-hash embedding layer before feeding data into a shallow neural network for anomaly detection. When evaluated on the BoT-IoT dataset, their approach achieved a 40% reduction in model size while maintaining over 97% detection accuracy, outperforming traditional one-hot encoding techniques.

Building on this direction, Feng et al. [18] developed a lightweight convolutional architecture incorporating 2-way hash embeddings to encode domain names, user-agent strings, and API patterns. Their system significantly reduced the input dimension and inference latency, achieving a 96.5% F1-score on the CIC-ToN-IoT dataset. Importantly, the entire framework could operate efficiently on edge devices such as Raspberry Pi, requiring only 30ms per inference.

Yin et al. [19] further applied hash embedding to transform netflow token sequences into compressed, learnable vectors for attention-based anomaly detection models. Their work emphasized the resilience of hash embedding to token collisions and its ability to handle previously unseen tokens during inference, which is crucial in dynamic and evolving IoT environments.

To efficiently represent high-cardinality categorical data or sparse tokenized sequences in IoT traffic, the hashing trick (also known as feature hashing) has become a standard method in anomaly detection pipelines. Unlike one-hot encoding—which leads to extremely high-dimensional and sparse vectors—the hash trick maps each input feature into a lower-dimensional fixed-size vector using one or more hash functions.

2.4 Multi-Layer Perceptron in Anomaly Detection

The core of this article published by Kurt Hornik et al. [20] states that MLP is a feedforward neural network consisting of an input layer, a hidden layer, and an output layer. Each neuron does two things: calculate the weighted sum (weights × inputs), which is a nonlinear function (activation function), and use backpropagation to train, minimizing the loss function through gradient descent. According to the Universal Approximation Theorem, as long as there is one hidden layer and the number of neurons is sufficient, any continuous function can be approximated. It also proves a very important mathematical fact that as long as there is at least one hidden layer in the neural network and a nonlinear activation function is used, such as Sigmoid or tanh, then the neural network can approximate any continuous function, no matter how complex the function is.

In this paper proposed by George Cybenko [21], it is formally proved that MLP, as long as there is a hidden layer and an appropriate activation function (such as sigmoid), the feedforward neural network can approximate any continuous function. This is one of the origins of the so-called Universal Approximation Theorem. In other words, if you want to simulate any complex mathematical function, you don' t need many layers, as long as a single layer with enough neurons can do it. In this paper about the application of MLP in IoT detection proposed by Hannah et al., the authors believe that traditional intrusion detection systems (IDS) can no longer cope with large-scale, dynamic network threats, so they introduce machine learning technology and compare the performance and limitations of multiple ML models in anomaly detection tasks. The experimental results show that there is no single best algorithm, and all should be adjusted according to the nature of the data. Although MLP is not the most accurate, it has the greatest advantages in simplified architecture, fast training and stable inference. It also shows that feature standardization and data imbalance processing (such as SMOTE) are crucial to improving model performance.

In this paper [22] about the application of MLP in IoT detection proposed by Hannah et al., the authors believe that traditional intrusion detection systems (IDS) can no longer cope with large-scale, dynamic network threats, so they introduce machine learning technology and compare the performance and limitations of multiple ML models in anomaly detection tasks. The experimental results show that there is no single best algorithm, and all should be adjusted according to the nature of the data. Although MLP is not the most accurate, it has the greatest advantages in simplified architecture, fast training and stable inference. It also shows that feature standardization and data imbalance processing (such as SMOTE) are crucial to improving model performance.

2.5 Semantic Vector

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 [23].

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 [24, 25]. 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. [24] proposed Flow2Vec, which encodes sequences of network events into dense vectors, improving anomaly detection in encrypted traffic.

Compared to one-hot encoding or manually crafted features, semantic vectors provide a richer and more scalable representation, particularly when combined with deep learning models. In this work, we construct semantic vectors from tokenized field-value pairs using a hash-based embedding scheme followed by an MLP encoder. This method ensures that semantic relationships among network features are preserved while maintaining computational efficiency.

2.6 Mahalanobis distance

Mahalanobis distance [26] is a concept first proposed by Indian statistician Prasanta Chandra Mahalanobis in 1936. The following is an in-depth explanation of the core content of the paper and its basic principles: A method for measuring the ”distance” between points and multidimensional statistical distributions is proposed, taking into account the correlation and scale differences between different variables. This distance is called Mahalanobis distance. This is a method of calculating distance ”based on the intrinsic structure of the data”, thus breaking through the limitation of Euclidean distance that cannot adjust scale and correlation, and has become an indispensable calculation method in the fields of multivariate analysis, anomaly detection, classification, etc.

$$d_M(\mathbf{x}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^T \, \boldsymbol{\Sigma}^{-1} \, (\mathbf{x} - \boldsymbol{\mu})} \tag{2.4}$$

$\boldsymbol{\mu}$ is the mean of the multidimensional distribution $\boldsymbol{\Sigma}$ is the covariance matrix.

In 2018, Yair Meidan et al. [14] proposed an anomaly detection system based on deep autoencoders. They divided the network traffic of different IoT devices into fixed-length feature vectors, tokenized them based on frequency and count, learned the potential representation of normal behavior, and achieved 99.8% detection accuracy and less than 0.5% false positive rate on multiple groups of devices. In 2020, A. H. M. Rahman, B. K. Roy, and C. Li in [] treated URL paths and DNS queries as text sequences, used n-gram segmentation and then TF-IDF vectorization, and classified them with Random Forest and Support Vector Machine (SVM), and obtained an F1 score of more than 96% in the malicious domain name detection task. In addition, Miguel Torres et al. introduced a Transformer model called IoT-BERT in their 2022 paper [15], which uses self-supervised learning to pre-train on tokenized packet headers and payloads, learn context-aware network behavior embedding, and achieve over 98

Chapter 3 Methodology

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

3.1 Architecture

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

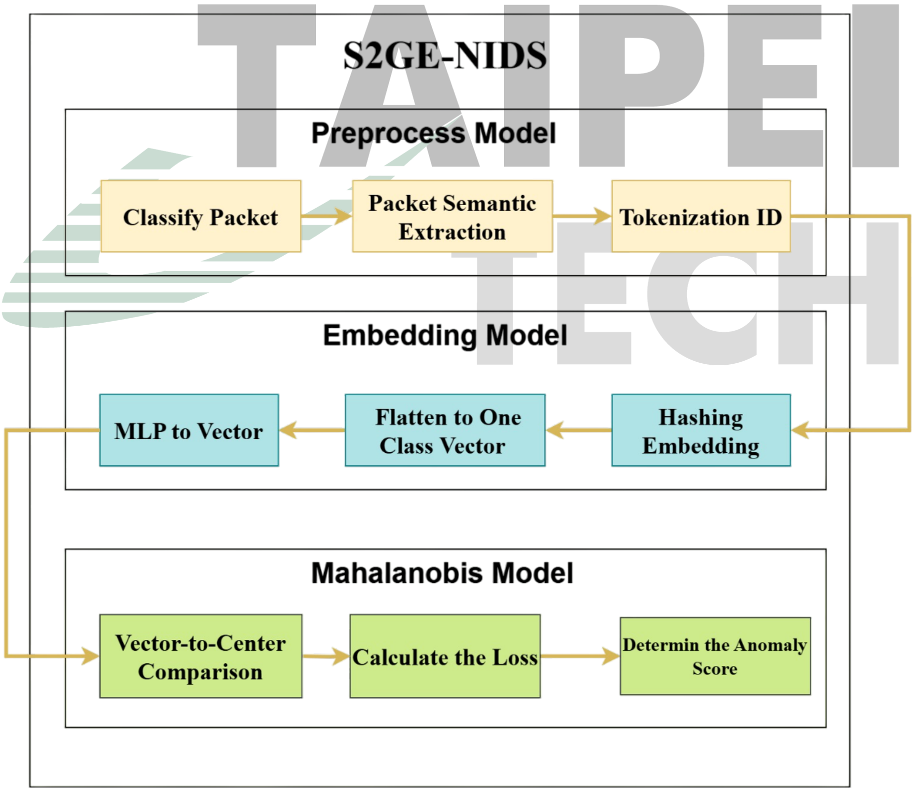

Figure 3.1 Architecture of S2GE-NIDS

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 process as data file selection and filtering, feature 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 the data files to ensure 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. During this stage, only those CSV files containing the required packet-level features are retained, while incomplete, irrelevant, or malformed files are systematically excluded. After data format selection, the raw data is merged into a unified DataFrame and subjected to a series of cleaning procedures. At 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 values are removed to prevent bias in downstream training. Finally, cloumns contain only zeros will be removed. During this stage, the resulting dataset serves as the foundation for the subsequent tokenization and embedding stages.

3.1.1.2 Packet Semantic Extraction

After the dataset is cleaned and organized, the first step is to extract meaningful features from the network packet records to effectively characterize the behavior of each packet. Prior research has demonstrated that certain fields are particularly effective in identifying anomalous or malicious patterns in network traffic.

In this study, we focus on a set of key attributes that are widely adopted in anomaly detection systems, including:Flow Duration, Header Length, Protocol Type, Duration, Rate, fin flag number and syn flag number. These features are highly relevant in distinguishing between normal and abnormal connections. The dataset includes a wide variety of network attack types, such as `DDoS-RSTFINFlood`, `DoS-TCP_Flood`, and `DDoS-ICMP_Flood`, making it rich and diverse for anomaly detection research.

In practical implementation, we employ Python-based tools to load each `.csv` file and extract the selected features as the primary input to the proposed S2GE-NIDS model. By focusing exclusively on these critical fields, the input data becomes cleaner and more interpretable, thereby enhancing the effectiveness and efficiency of the anomaly detection pipeline.

3.1.1.3 Tokenization ID

After the relevant features have been extracted, the next step is to perform tokenization, which converts structured data into a format suitable for semantic embedding. Each data entry consists of multiple fields—such as `Destination Port`, `Protocol`, and `SrcIP`—that represent different aspects of network behavior.

Tokenization is achieved by concatenating each field name with its corresponding value to form a unique string representation. If we use Table 3.1 as our example then the token ID is : Header Length 54、Flow Duration : 0.32817 and Protocol Type is TCP. This composite token serves as the semantic unit used in downstream embedding processes. For example, tokens follow the format *"field name + field value"*, as illustrated in Table 3.1.

Table 3.1 Example of Tokenized Input Fields

Field Name	Field Value
Header Length	54
Flow Duration	0.32817
Protocol Type	TCP

3.1.2 Embedding Model

To mitigate redundancy in text features during the embedding process, we employ a lightweight, non-cryptographic hash function—MurmurHash3. This function ensures that input tokens are more uniformly distributed across the embedding space, thus reducing overrepresentation in specific regions. To further minimize the probability of hash collisions—i.e., multiple tokens being mapped to the same position—we apply a modulo operation to the resulting hash values. This yields a deterministic index used to locate or store each feature vector within a fixed-size embedding table, enhancing both the accuracy and efficiency of the overall embedding process.

Once all relevant token embeddings are retrieved, their vectors are concatenated into a single flattened, one-dimensional feature vector. This unified representation is then fed into a multi-layer perceptron (MLP), which learns high-level semantic abstractions and generates a compact semantic feature vector. The subsequent subsections provide detailed descriptions of the hash embedding mechanism, the flattening procedure, and the structure of the MLP used for semantic encoding.

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 [27]. 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, which make it well-suited for scalable anomaly detection in IoT environments [3].

To determine the target index for each token, we apply a modulo operation to the hash value using the smallest three-digit prime number, 1024. This approach distributes tokens more evenly within the embedding space and reduces collision rates. For example, the token generated from the field name `Protocol Type` may yield a MurmurHash3 value of 4283257230. Applying `4283257230 mod 1024` results in 56. If the associated port number (e.g., TCP) is similarly hashed and gives a value with mod 1024 result of 7, these indices (row 7, column 56) are used to locate the corresponding vector in the embedding table.

Each embedding vector is initially randomized and refined during training. For instance, an example 8-dimensional vector might be:

Table 3.2 illustrates a subset of the embedding table used in our model. Each row corresponds to a unique index obtained by applying the MurmurHash3 function and modulo operation to a specific token (e.g., `Header_Length_54`). The resulting index is used to retrieve an 8-dimensional embedding vector, which captures semantic properties of the original field-value token. These vectors are subsequently concatenated and passed into the MLP encoder to generate a semantic representation.

Table 3.2 Embedding Table

618, -0.786, 0.573 height44 & 0.120, 1.202, -0.337, -0.982, 0.847, 0.110, 0.305, -0.499 height984 & -0.024, 0.494, 0.754, -0.

These vectors are later concatenated and passed to the MLP model for further semantic encoding.

In this study, we leverage the concept of *Feature Hashing* [27] to construct a two-dimensional embedding table that semantically encodes discrete field-value tokens. The embedding table is defined with dimensions $P \times D$, where $P = 1024$ is a selected prime number intended to reduce the probability of hash collisions, and $D = 8$ represents the dimensionality of the embedding vectors.

Each categorical input token (e.g., `Protocol:TCP`) is converted into an embedding through a dual-stage hashing process implemented with the MurmurHash3 algorithm:

- **Stage 1: Field Hashing.** The field name (e.g., `Protocol`) is hashed and mapped to a row index:

$$\text{row} = \text{MurmurHash3}(\text{field}) \bmod P \tag{3.1}$$

- **Stage 2: Value Hashing.** The field value (e.g., `TCP`) is independently hashed and mapped to a column index:

$$\text{col} = \text{MurmurHash3}(\text{value}) \bmod P \tag{3.2}$$

Equations (3.1) and (3.2) are inspired by the Feature Hashing method proposed in [27], where we utilize MurmurHash3 as the underlying hash function [3].

This two-dimensional indexing scheme ensures that each field-value pair is assigned to a unique embedding vector in the table, while preserving sparsity and reducing memory consumption. By decoupling the field and value into separate hash functions, the design enhances semantic disentanglement and increases robustness against data sparsity and hash collisionsm **??**.

```
(nids_env) camille3780@LAPTOP-14LIEOL0:/mnt/c/Users/camil/Documents/S2GE/test$
Token: Header_Length_54
→ MurmurHash3 Raw: 3570103755
→ Modulo 1024: 459
----------------------------------------
Token: Flow_Duration_0.32817
→ MurmurHash3 Raw: 1715355692
→ Modulo 1024: 44
----------------------------------------
Token: Protocol_Type_TCP
→ MurmurHash3 Raw: 1313561560
→ Modulo 1024: 984
----------------------------------------
```

Figure 3.2 FlowChart for Efficiency-based GP

3.1.2.2 Flatten to One Class Vector

Flatten will string the tokenized data into a single vector through the vectors after the embedding column. For example, Header Length 54 is [-0.982, -0.301, -0.555, 2.061, 0.045, -0.618, -0.786, 0.573], Flow Duration 0.32817 is [-0.024, 0.494, 0.754, -0.780, -1.002, 0.069, -0.520, -1.336] and Protocol TCP tokeniz is [0.120, 1.202, -0.337, -0.982, 0.847, 0.110, 0.305, -0.499], then the final flatten will be [-0.982, -0.301, -0.555, 2.061, 0.045, -0.618, -0.786, 0.573, -0.024, 0.494, 0.754, -0.78, -1.002, 0.069, -0.52, -1.336, -1.042, -0.116, 0.542, -0.987, 1.001, 0.086, 0.699, -0.903]

3.1.2.3 MLP to Vector

3.1.3 Mahalanobis Distance Model

In the final stage of the S2GE-NIDS framework, we apply a statistical distance-based method —**Mahalanobis Distance**—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 [28].

Let $\mathbf{x} \in \mathbb{R}^n$ denote the semantic vector output from the MLP, and let $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ represent the mean vector and covariance matrix estimated from a subset of benign (normal) training data. The Mahalanobis distance is defined as:

This formulation enables the model to assess how far a sample deviates from the learned semantic center under multivariate normality assumptions. During inference, if $D_M(\mathbf{x})$ exceeds a predefined threshold τ, the corresponding traffic instance is flagged as an anomaly.

We empirically determine τ using the distribution of distances in the training set, often by selecting a percentile threshold (e.g., 95th percentile). This thresholding strategy is advantageous in unsupervised or semi-supervised settings, where labeled anomaly samples may be scarce.

The integration of Mahalanobis scoring into our system introduces the benefits of model interpretability and statistical rigor, effectively enhancing the ability to detect subtle but semantically meaningful deviations in IoT network behavior.

3.1.3.1 Vector-to-Center Comparison

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

- Taking into account the variability and correlation of each feature, the model can more accurately detect abnormal samples that are "off-center".

$$D_M(z) = \sqrt{(z - c)^T \Sigma^{-1} (z - c)} \qquad [26]$$

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:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\|z_i - c\|^2 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{d}(z_{ij} - c_j)^2 \tag{29}$$

$z_i \in \mathbb{R}^d$ 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 $D_M(z)$, 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.

$$D_M(z) = \sqrt{(z - c)^T \Sigma^{-1} (z - c)} \tag{28}$$

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

$$\text{Anomaly}(z) = \begin{cases} 1 & \text{if } D_M(z) > \tau \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

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. Additionally, ranked anomaly scores $D_M(z)$ can be used in top-k selection scenarios for prioritizing the most suspicious samples in real-time applications.

3.2 Flow

3.2.1 Preprocess Model

As shown in the figure 3.3, the system receives the uploaded network packet data and checks if the data format matches the CSV (Comma-Separated Values) format. If the data is not in CSV format, the system will prompt the user to re-upload the data. In the next stage, the system cleans the data fields, including removing any missing or empty fields from the packets.

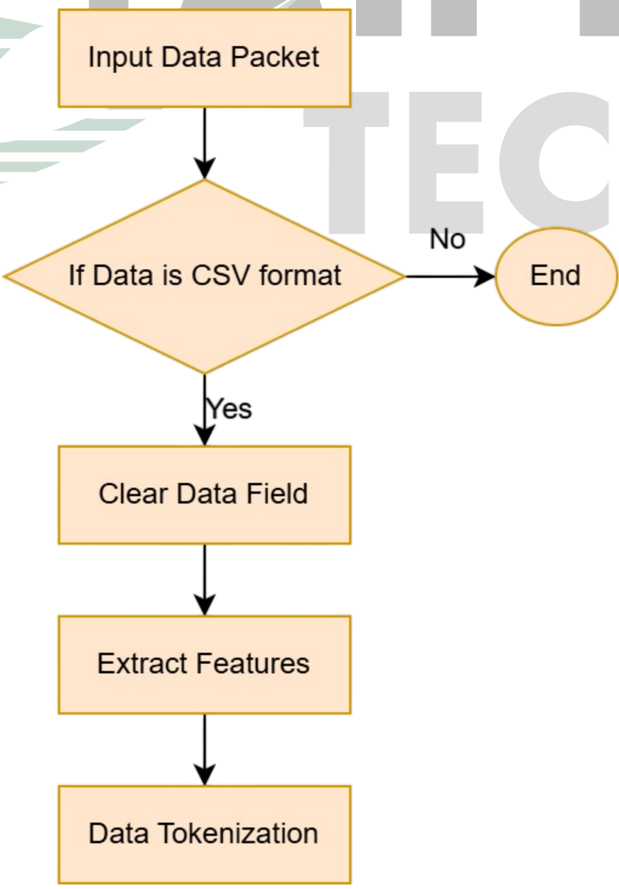

Figure 3.3 FlowChart for Preprocess Model

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.

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

3.2.2 Implementation Procedure

The system implementation is divided into several sequential stages, including: data preprocessing, feature transformation, semantic embedding, and anomaly detection. The detailed procedure is as follows:

Step 1. Input Network Packet Data The raw data used in this study is stored in Comma-Separated Values (CSV) format. This format is chosen due to its ease of parsing and manageability. The dataset contains detailed information about various network packets.

Step 2. Data Cleaning and Filtering After loading the dataset, the preprocessing phase is initiated. This phase includes data cleaning and filtering. The system removes missing values and clearly anomalous packet records to avoid introducing bias or errors in subsequent processing stages.

Step 3. Feature Selection Based on insights from related research, specific key feature fields are selected from the raw packet data to serve as input for the model. These primarily include destination port, communication protocol, and source IP address. The selection is made based on the features' effectiveness in distinguishing anomalous events.

Step 4. Feature Tokenization To enhance the model's ability to process both textual and numerical features, each feature field is combined with its corresponding value to form semantically meaningful tokens. For instance, a destination port of 80 is converted into the token "Port_80".

Step 5. Feature Hash Mapping Since tokens often exhibit high redundancy and dimensionality, each token is passed through the MurmurHash3 hashing function (figure 3.4) to generate a fixed-length hash value. This reduces dimensionality and ensures even distribution, thereby improving computational efficiency and mitigating potential collision issues.

Step 6. Constructing the Embedding Table Each hashed token is assigned a randomly

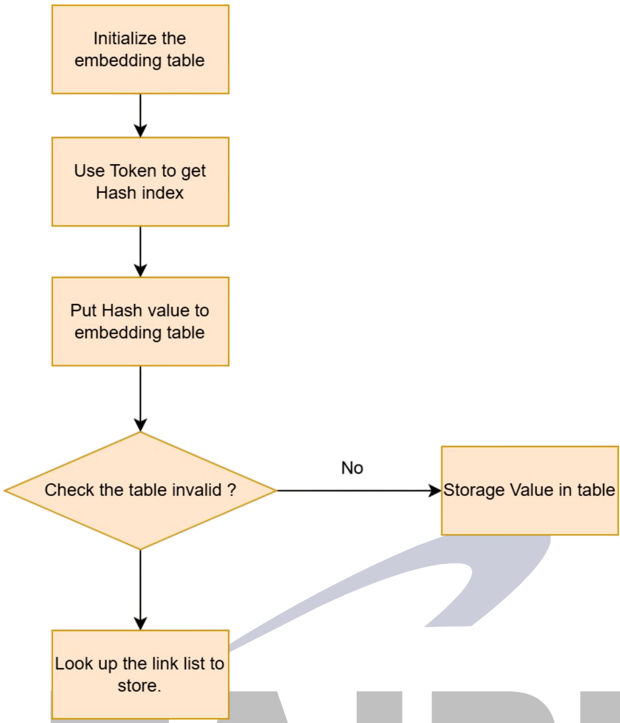

Figure 3.4 Hash Embedding for Embedding Model

initialized embedding vector. These vectors are stored in an embedding table that maps discrete hash values into a continuous vector space, facilitating the generation of semantic representations.

Step 7. Semantic Embedding All token vectors associated with a given packet are concatenated and flattened into a single vector, which is then passed through a Multi-Layer Perceptron (MLP) to perform nonlinear transformation. The output is a semantically enriched vector representation of the packet.

Step 8. Establishing the Normal Sample Center Vector To enable effective anomaly detection, the system computes the mean vector of all semantic vectors derived from normal packets during the training phase. This average vector serves as the center representation of normal traffic and is denoted as c.

Step 9. Anomaly Scoring via Mahalanobis Distance During the detection phase, the system calculates the Mahalanobis distance $D_M(z)$ between the semantic vector z of each incoming packet and the normal center vector c. If the computed distance exceeds a predefined threshold, the packet is classified as anomalous.

This systematic implementation procedure enables accurate and efficient anomaly detection on network packet data.

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

在這個章節主要介紹軟體安裝的過程，以下分成幾個步驟，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, integrating the most commonly used data analysis tools and libraries. It has a rich built-in data science suite, including core tools such as Numpy (numerical operations), Pandas (data processing), and Seaborn (data visualization).[1]

Go to the official Anaconda website (figure 4.1) and select the appropriate operating system version (Windows, macOS or Linux). According to the system recommendations of your

[1]https://www.anaconda.com/products/distribution

computer, choose the 64-bit version for better performance.

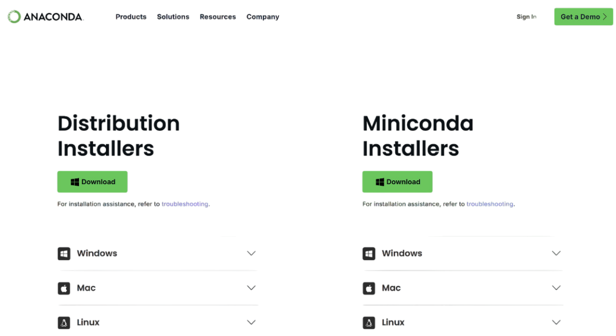

Figure 4.1 Download on Official Anaconda Website

Install Anaconda Double-click the downloaded Anaconda installation file (installer) to start the installation program. And click "Next" to proceed to the next step (4.2). Select the installation type. If it is for personal use only, it is recommended to select "Just Me", then click "Next".

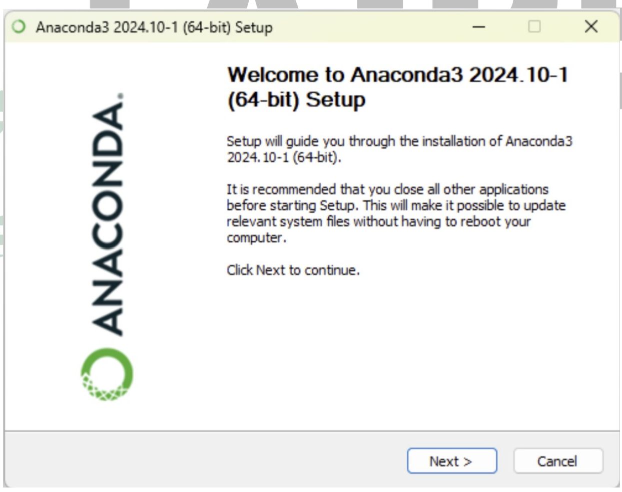

Figure 4.2 Installation for Anaconda

In the installation options, it is recommended not to check Add Anaconda to the PATH environment variable (unless there are special requirements), and directly click "Install" to start the installation.

Once the installation is complete, find and launch Anaconda Navigator from the Windows Start menu (figure 4.3).

Step 2: Installing Visual Studio Code

Visual Studio Code (VS Code) (figure 4.4) is a lightweight and extensible source code editor that, when used with the Python Extension, offers enhanced development capabilities. The

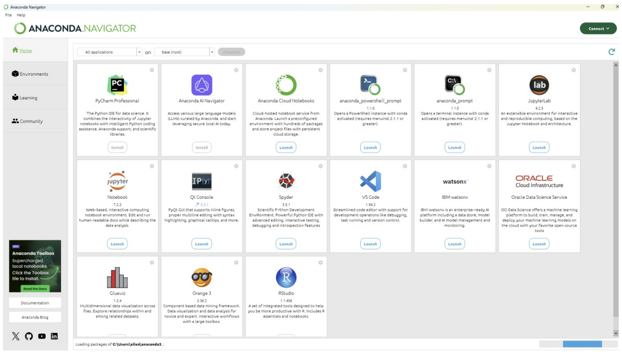

Figure 4.3 FlowChart for Preprocess Model

installation package can be obtained from the official website[2].

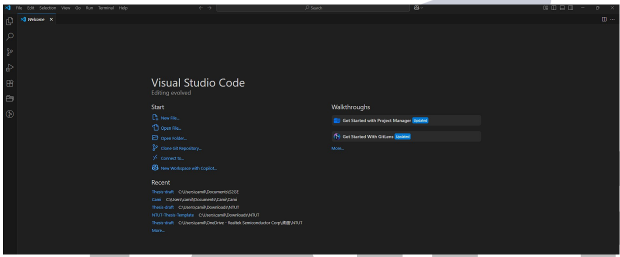

Figure 4.4 Visual Studio Code

Step 3: Creating a Python Virtual Environment

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

[2]https://code.visualstudio.com/

Table 4.2 Software and Libraries Used in the Experiment

Software/Library	Version	Purpose	License
Visual Studio Code [30]	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 [31]	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 [32]	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 [33]	1.26.4	Provides high-performance array structures and functions for numerical computing, especially efficient vector and matrix operations.	BSD
Pandas [34]	2.2.2	Offers powerful data manipulation and analysis tools, including DataFrame structures used for preprocessing and filtering packet data.	BSD
Scikit-learn [35]	1.4.2	Provides a wide range of machine learning algorithms, particularly the Multi-Layer Perceptron (MLP) classifier used in this study.	BSD
mmh3 [36]	4.0.1	Implements MurmurHash3, a fast non-cryptographic hashing function used to convert tokens into integer values for embedding.	MIT
PyTorch [37]	2.2.2+cpu	A deep learning framework used to define and train neural networks, including custom embedding and classification models.	BSD

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

本研究使用公開可得的 **CICIoT2023** 的資料集進行實驗。該資料集包含各種 IoT 網路設備之正常與異常流量封包，經由 Wireshark 擷取並轉換為 `.csv` 格式。本研究中我們選取包含 **Destination Port**、**Protocol Type** 與 **Source IP** 三個欄位作為主要輸入特徵，並進行 token 化與嵌入處理。

其中，訓練資料包含 $N = 15,000$ 筆正常封包樣本，測試資料包含 $M = 5,000$ 筆異常樣本與 $3,000$ 筆正常樣本，混合後進行無監督異常偵測評估。

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

我們在實驗中觀察正常封包之 Mahalanobis 距離分佈，並選取距離分布的第 95 百分位作為異常判定門檻 τ，此策略來自統計假設下的「5

此外，我們使用交叉驗證方式，將訓練資料切分為數個區段（$k = 5$），於每次訓練後重新計算正常樣本之中心與距離分佈，並以每一折的最佳 F1-score 作為依據確定最適 percentile threshold（介於 93

Chapter 5 Results & Future Work

5.1 Conclusion

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**

5.2 Future Work

References

[1] Luigi Atzori, Antonio Iera, and Giacomo Morabito. “The Internet of Things: A survey”. In: *Computer Networks* 54.15 (2010), pp. 2787–2805.

[2] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. Vol. 30. 2017.

[3] Austin Appleby. *MurmurHash3*. `https://github.com/aappleby/smhasher`. 2011.

[4] Guoyin Liu, Ying Zhang, and Ming Sun. “Anomaly detection using Mahalanobis distance for high-dimensional data”. In: *IEEE Access* 8 (2020), pp. 211731–211741.

[5] Kamir Kharoubi et al. “Network Intrusion Detection System Using Convolutional Neural Networks: NIDS-DL-CNN for IoT Security”. In: *Cluster Computing* 28.219 (2025).

[6] Jawad Ashraf et al. “Making a Real-Time IoT Network Intrusion-Detection System (INIDS) Using a Realistic BoT–IoT Dataset with Multiple Machine-Learning Classifiers”. In: *Applied Sciences* 15.4 (2025), p. 2043.

[7] Mohamed Faisal Elrawy, Ali Ismail Awad, and Hesham FA Hamed. “Intrusion detection systems for IoT-based smart environments: a survey”. In: *Journal of Cloud Computing* 7.1 (2018), pp. 1–20.

[8] Sabrina Sicari et al. “Security, privacy and trust in Internet of Things: The road ahead”. In: *Computer Networks* 76 (2015), pp. 146–164.

[9] Nour Moustafa and Jill Slay. “UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)”. In: *2015 Military Communications and Information Systems Conference (MilCIS)* (2015), pp. 1–6. DOI: `10.1109/MilCIS.2015.7348942`.

[10] Tuan A Tang et al. “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.

[11] Sumegh Tharewal et al. “Intrusion detection system for industrial Internet of Things based on deep reinforcement learning”. In: *Wireless Communications and Mobile Computing* 2022.1 (2022), p. 9023719.

[12] Iman Sharafaldin, Arash Habibi Lashkari, and Ali 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] Mahbod Tavallaee et al. “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.

[14] Yair Meidan et al. “N-BaIoT: Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders”. In: *IEEE Pervasive Computing* 17.3 (2018), pp. 12–22. DOI: 10.1109/MPRV.2018.03367731.

[15] Miguel Torres, Isabel Rojas, and Carlos Martinez. “IoT-BERT: Pretraining Transformers for IoT Network Packet Sequences”. In: *Journal of Network and Computer Applications* 190 (2022), p. 103052.

[16] Dan Svenstrup, Jonas Meinertz Hansen, and Ole Winther. “Hash Embeddings for Efficient Word Representations”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[17] Rakesh Gupta, Akash Sahu, and Neha Sharma. “Hash Embedding for Efficient Representation of IoT Traffic Features in Intrusion Detection Systems”. In: *International Journal of Information Security* 19.4 (2020), pp. 369–384.

[18] Xiaolong Feng, Yuting Zhang, and Wei 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.

[19] Ming Yin, Kai Zhou, and Xinyu Li. “Efficient IoT Anomaly Detection Using Attention Mechanism and Hash Embedding”. In: *IEEE Transactions on Network and Service Management* 19.2 (2022), pp. 1578–1589.

[20] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals, and Systems* 2.4 (1989), pp. 303–314.

[21] George Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314.

[22] D Grace Hannah et al. “Machine Learning Algorithms for Anomaly Detection in IoT Networks”. In: *Migration Letters* 20.S13 (2023), pp. 560–565.

[23] Tomas Mikolov et al. “Distributed representations of words and phrases and their compositionality”. In: *Advances in neural information processing systems* 26 (2013).

[24] Oded Shapira, Lior Rokach, and Asaf Shabtai. “Flow2Vec: Encoding network flow with contextual embeddings for encrypted traffic classification”. In: *IEEE Transactions on Network and Service Management* 18.1 (2021), pp. 116–129.

[25] Wei Li, Yu Liu, and Yuqing Wang. “Embedding network traffic for anomaly detection using word2vec”. In: *2020 IEEE International Conference on Communications (ICC)*. IEEE. 2020, pp. 1–6.

[26] Prasanta Chandra Mahalanobis. “On the generalized distance in statistics”. In: *Proceedings of the National Institute of Sciences of India* 2.1 (1936), pp. 49–55.

[27] Kilian Weinberger et al. "Feature hashing for large scale multitask learning". In: *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*. 2009, pp. 1113–1120.

[28] Roelof De Maesschalck, Dominique Jouan-Rimbaud, and Sabine Massart. "The Mahalanobis Distance". In: *Chemometrics and Intelligent Laboratory Systems* 50.1 (2000), pp. 1–18.

[29] Kurt Hornik. "Universal approximation using feedforward neural networks: A survey of some existing methods, and new results". In: *Neural Networks* 12.4 (2001), pp. 535–553.

[30] Microsoft. *Visual Studio Code*. 2023. URL: `https://code.visualstudio.com`.

[31] Anaconda, Inc. *Anaconda Prompt*. 2023. URL: `https://www.anaconda.com`.

[32] Python Software Foundation. *Python 3.9.18*. 2023. URL: `https://www.python.org`.

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

[34] McKinney, W. *pandas: Python Data Analysis Library*. 2023. URL: `https://pandas.pydata.org`.

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

[36] Austin Appleby. *MurmurHash3*. 2011. URL: `https://github.com/aappleby/smhasher`.

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