University of Massachusetts Dartmouth

Explainability of Network Intrusion Detection using Transformers: A Packet-Level Approach

A Thesis in

Data Science

by

Pahalavan Rajkumar Dheivanayahi

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

August 2023

We approve the thesis of Pahalavan Rajkumar Dheivanayahi

Date of Signature

Gokhan Kul
Assistant Professor, Department of Computer and Information Science
Thesis Advisor

Bharatendra Rai
Professor, Department of Decision and Information Sciences
Thesis Committee

Lance Fiondella
Associate Professor, Department of Electrical and Computer Engineering
Thesis Committee

Scott Field
Associate Professor, Department of Mathematics
Graduate Program Director, Data Science
Thesis Committee

Jean VanderGheynst
Dean, College of Engineering

Tesfay Meressi
Associate Provost for Graduate Studies

Abstract

Explainability of Network Intrusion Detection using Transformers: A Packet-Level Approach

by Pahalavan Rajkumar Dheivanayahi

Network Intrusion Detection Systems (NIDS) are critical in ensuring the security of connected computer systems by actively detecting and preventing unauthorized activities and malicious attacks. Machine learning (ML) and deep learning (DL) based NIDS models leverage algorithms that learn from historical network traffic data to identify patterns and anomalies to capture complex relationships. The primary objective of this research is to generate tags and descriptions for the packets that are difficult to classify by the NIDS. Most NIDS datasets that are publicly available have focused on flow data, offering aggregated information about network connections, and have played a crucial role in enabling researchers and network security professionals to design and develop flow-based NIDS solutions. While flow records provide valuable information for detecting network-level anomalies and attacks, they do not consider packet-level information and payload contents. In this research, we propose a packet-level approach for NIDS that leverages the flow information with the packet header fields and payload. To facilitate this research, we have curated a comprehensive Packet-level dataset constructed by extracting the Packet Capture (PCAP) files from two widely used flow-level datasets, namely CIC-IDS2017 and UNSW-NB15. Recent advancements in Natural Language Processing (NLP) have demonstrated the effectiveness of Transformer-based models in handling sequence data with tasks such as token classification and text generation. We have adapted this technology to NIDS to extract key features and characteristics of the header and payload in the context of various attacks. Unlike traditional classification methods that assign predefined labels to network packets, this method focuses on generating tags based on the packet signature that explains the packet content and potential risks. The tags and descriptions offer network security professionals a tool to comprehend suspicious packets with an unfamiliar or potentially malicious signature, assess their nature, and help make informed decisions promptly.

Acknowledgments

First and foremost, my sincere appreciation goes to my advisor, Dr. Gokhan Kul. His unwavering support, insightful guidance, valuable suggestions, and unending patience have been instrumental in shaping this research. His mentorship has been invaluable, and I am truly grateful for his constant encouragement and trust in my abilities.

I thank Dr. Nathaniel D. Bastian of the United States Military Academy (USMA) at West Point for funding this project. His financial support and intellectual input have been pivotal in carrying out this research, and I sincerely appreciate his thoughtful feedback and ideas that have enriched the work.

I extend my heartfelt thanks to my thesis committee members for their willingness to serve and provide valuable insights.

I thank Dr. Bharatendra Rai for his inspiring Business Analytics and Data Mining class, which kindled my interest in machine learning topics and practical applications.

I am grateful to Dr. Lance Fiondella for his input and suggestions that have contributed to improving this research.

I acknowledge Dr. Scott Field for his generous assistance in setting up the Tukey Server and for helping me resolve technical challenges.

Lastly, I sincerely thank the Data Science program and the Center for Scientific Computing and Data Science Research for providing access to the Tukey Server. Its powerful resources, including 64 AMD Epyc cores, 1 TB of DDR3 RAM, 20 TB of storage, and two NVIDIA A100 GPUs with 80 GB of RAM, have been indispensable in meeting the computational demands of this research.

Table of Contents

List of Figures

List of Tables

Abbreviations

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CBOW	Continuous Bag of Words
CSV	Comma Separated Values
DDoS	Distributed Denial of Service
DoS	Denial of Service
FTP	File Transfer Protocol
GPT	Generative Pre-Trained Transformer
GPU	Graphics Processing Unit
GloVe	Global Vectors for Word Representation
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LSTM	Long Short-Term Memory
LLM	Large Language Model
LoRA	Low-Rank Adapters
NIDS	Network Intrusion Detection System
NLP	Natural Language Processing
OSI	Open Systems Interconnection
PCAP	Packet Capture
PEFT	Parameter-Efficient Fine-Tuning
PLM	Pre-trained Language Models
QLoRA	Quantized Low-Rank Adapters
RNN	Recurrent Neural Network
SQL	Structured Query Language
SSH	Secure Socket Shell
TCP	Transmission Control Protocol
t-SNE	t-Distributed Stochastic Neighbor Embedding

Chapter 1: Introduction

In today's digital age, where technology changes every aspect of our lives, network security plays a critical role in ensuring the confidentiality, integrity, and availability of information. A network intrusion detection system (NIDS) is a security tool designed to monitor network traffic and detect any unauthorized or suspicious activities within a computer network. Its primary purpose is to identify and respond to potential security breaches, attacks, or policy violations in real time. Network intrusion detection systems employ various techniques to identify and respond to potential intrusions [1]. These techniques can be broadly classified into two categories namely Signature-based detection [2], [3] and Anomaly-based detection [4], [5]. Signature-based detection relies on known patterns or signatures of previously identified attacks, as shown in Figure 1. The system compares network traffic against an extensive database of attack signatures to detect and block known threats. On the other hand, Anomaly-based detection involves establishing a baseline of normal network behavior and identifying deviations from this baseline. By monitoring network traffic statistics [6], protocol usage, and resource utilization [7], anomaly detection algorithms can identify unusual activities that *may* indicate an intrusion.

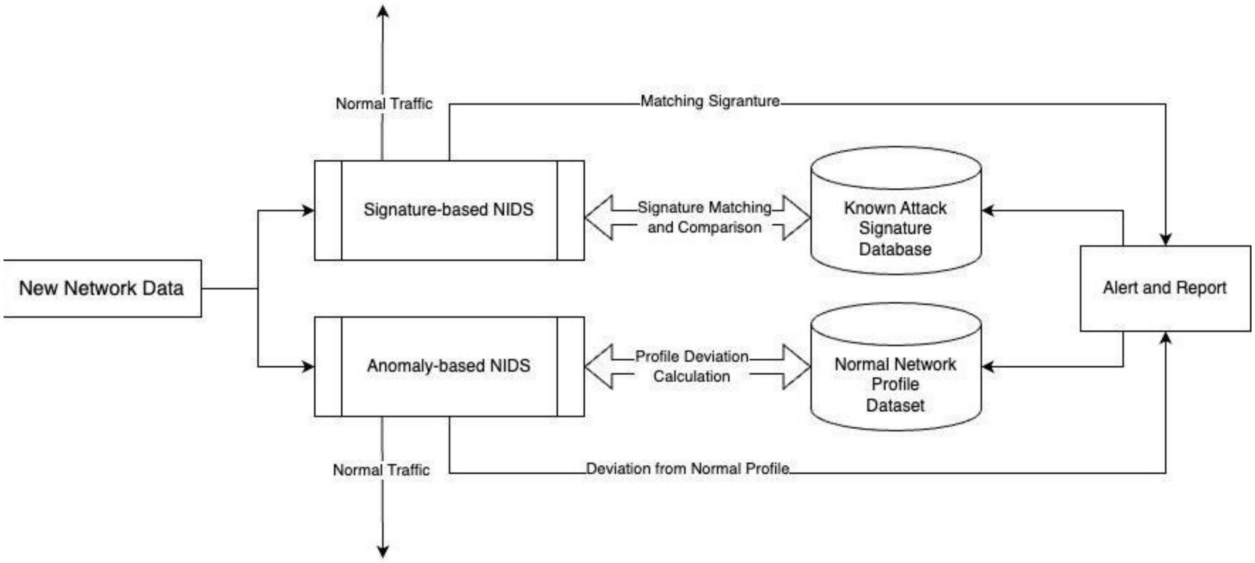

Normal Traffic
Matching Sigranture
Signature-based NIDS
Signature Matching and Comparison
Known Attack Signature Database
New Network Data
Alert and Report
Anomaly-based NIDS
Profile Deviation Calculation
Normal Network Profile Dataset
Normal Traffic
Deviation from Normal Profile

Figure 1: Workflow of Signature-based and Anomaly-based Intrusion Detection System

Flow-based Network Intrusion Detection System (NIDS) is a type of intrusion detection system that focuses on analyzing network flows to identify potential security threats and attacks [8], [9]. Network flows refer to the sequence of packet movement between a source and destination, characterized by attributes such as source and destination IP addresses, port numbers, protocol types, and timestamps. Flow-based NIDS operates by collecting and processing flow data generated by network devices, such as routers and switches, instead of analyzing individual packets. Flow-based NIDS is anomaly-based detection that leverages statistical analysis and machine learning algorithms to establish baselines of normal network behavior [10]. Any significant deviation from these baselines is flagged as a potential intrusion or security breach. CIC-IDS2017 [11], UNSW-NB15 [12], CTU-13 [13], and ISCX NSL-KDD [14] are some of the publicly available datasets that have been used for flow-based NIDS research and evaluation.

Signature-based intrusion detection, primarily using packet-level information, identifies network anomalies by comparing packet-level characteristics against predefined signatures or patterns. In signature-based detection, known attack patterns or signatures are derived from previously observed malicious activities. These signatures represent specific sequences or patterns of packets associated with a particular attack or exploit. When packet-level information matches a known signature, it indicates the presence of a known attack, triggering an alert or response. Packet-level information, including the packet header fields and payload, has been extracted using the PCAP files associated with the CIC-IDS2107 and UNSW-NB15 datasets and used for packet-level signature-based NIDS in this work.

Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) architectures have proven to be valuable in Network Intrusion Detection Systems (NIDS) for analyzing sequential data such as packet payload [15]. LSTMs introduced memory cells that can selectively remember or forget information over time [16]. This enables LSTMs to capture long-term dependencies in sequential data, effectively recognizing complex attack patterns. Transformer-based models use the encoder-decoder structure that leverages attention mechanisms to focus on relevant parts of the input sequence, providing more fine-grained analysis and selective information processing.

Transformers can benefit from transfer learning, where models pre-trained on large-scale datasets can be fine-tuned for specific NIDS tasks. Transformers process the entire sequence in parallel rather than sequentially, like RNNs. This parallelization allows for more efficient computation, making Transformers suitable for handling longer sequences and scaling to larger datasets [17], [18]. We have utilized BERT [19] and Falcon [20] models in this research for fine-tuning the packet-level information to identify and detect network attacks. BERT utilizes the encoder component of the transformer architecture to process input data bidirectionally, capturing contextual information from both left and right contexts. This bidirectional processing is beneficial for tasks that require a deep understanding of context, such as question answering, named entity recognition, and sentiment analysis. Falcon employs a decoder-only model, which is well-suited for text-generation tasks such as chat completion and translation. In the encoder-decoder architecture, the encoder is responsible for capturing contextual information and representing the input data in a lower-dimensional space where the decoder takes the encoded information and generates output sequences, reconstructing the original data format.

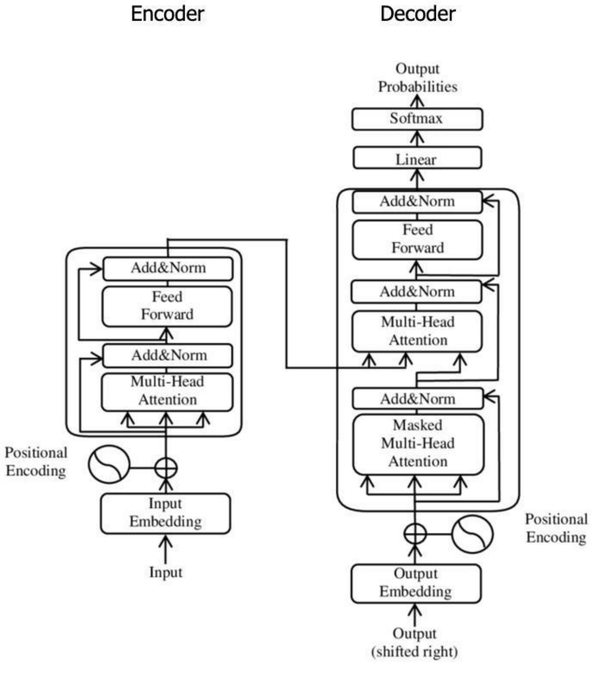


Figure 2: Encoder-Decoder Architecture of Transformer Model

The main objective of this research paper is the generation of tags for network packets using the CIC-IDS2017 and UNSW-NB15 datasets. In the dataset preparation phase, information is extracted and processed from the associated PCAP files, resulting in three subsets of datasets: Payload Bytes, Packet Bytes, and Packet Fields. The Payload Bytes dataset contains the decimal representation (ranging from 0 to 255) of the payload bytes of each packet. This subset enables detailed analysis of payload signature. The Packet Bytes dataset includes the decimal representation of the entire packet, providing a broader view of packet characteristics. The Packet Fields dataset comprises header fields and their corresponding values for all packets. This dataset offers valuable information about various packet attributes such as source and destination IP addresses, port numbers, protocol types, flags, and options. The flow_id column serves as a common identifier across all three subsets, facilitating the retrieval of complete packet information associated with each packet by joining the datasets. The original flow dataset is preprocessed, and the flow_id column is added to all packets in the three subsets. This is used to combine flow-level information with packet-level details. To enable easy utilization of these datasets in the future, a Python package has been developed. This package provides user-friendly functions for effortless access and usage of the datasets.

For tag generation, this paper takes a novel approach by leveraging the embeddings generated by the transformer model to assign tags to individual packets. The use of embeddings allows for a more nuanced representation of the packets in the embedding space. By clustering these embeddings, the paper aims to identify distinct groups or categories of packets. The resulting clusters can then be used to assign relevant tags to each packet, providing a descriptive label that goes beyond simple class predictions. One of the advantages of this approach is its ability to handle packets that may belong to new or previously unseen categories. By utilizing clustering techniques, the paper enables the model to assign tags to packets that fall outside the predefined classes, allowing for more comprehensive coverage and adaptability to evolving network conditions.

By associating packets with descriptive tags, network administrators and analysts can gain insights into the nature of network traffic, identify potential anomalies or threats, and make informed decisions regarding network management, security, and performance optimization. In addition to the tag generation process, we have also fine-tuned a text generation model using the Falcon model. This text generation model is designed to generate descriptive sentences about the packet header fields and the content of the payload. Through fine-tuning, we have enhanced the model's ability to generate accurate and informative descriptions that provide a deeper understanding of the network packet data. Moreover, we have fine-tuned the Falcon model to generate explanations for the generated tags. This enables the model to provide detailed insights and context regarding the behavior and characteristics of the network packets based on the generated tags. By combining tag generation and text generation capabilities, we have created a comprehensive framework that leverages transformer models and natural language processing techniques to extract meaningful information from network packet data. The utilization of open-source transformer models, such as BERT and Falcon, demonstrates the accessibility and applicability of these advanced models in the field of network security and intrusion detection to extract valuable insights for network analysis and defense.

Chapter 2: Dataset Preparation

2.1 Data Collection

In this research, we utilize two publicly available datasets, namely the CIC-IDS2017 and UNSW-NB15 datasets, to construct the dataset. These datasets are widely used in the field of network security and provide a rich collection of network traffic data for analysis and evaluation purposes. The data extraction process involves parsing and extracting packet-level information from PCAP (Packet Capture) files. PCAP files store network traffic data captured during network monitoring or analysis. By incorporating the CIC-IDS2017 and UNSW-NB15 datasets, the research benefits from the diverse and realistic nature of the included network traffic data.

2.1.1 CIC-IDS2017

The Intrusion Detection Evaluation Dataset (CIC-IDS2017) is a widely used dataset in the field of network security for evaluating the performance of intrusion detection systems (IDS). It was created by the Canadian Institute for Cybersecurity (CIC) and contains a comprehensive collection of network traffic data with various types of attacks [11]. The CIC-IDS2017 dataset is created by capturing real network traffic in a controlled environment, simulating both normal and malicious activities. The network traffic is generated by using a combination of different tools, including Metasploit, Hping3, and DDoSim, to create a diverse set of attacks and normal traffic patterns. The dataset provides over 80 network flow features, which are derived from the network traffic data using the CICFlowMeter tool. These features encompass a wide range of characteristics and attributes related to network flows, such as source and destination IP addresses, source and destination ports, protocol types, flow durations, packet counts, statistical features extracted from the traffic, and more. The dataset is divided into eight CSV files, each representing a specific period of the day. In addition to the CSV files, the CIC-IDS2017 dataset also provides access to the raw PCAP files. These files contain the original packet-level data captured during the network traffic collection process.

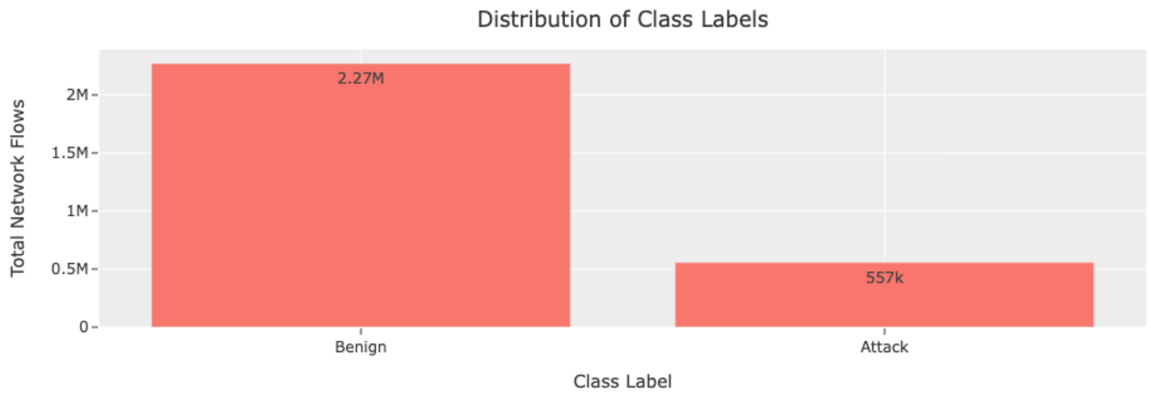


Figure 3: Distribution of Class Labels in the CIC-IDS2017 Dataset

The CIC-IDS2017 dataset provides 15 class labels that are assigned to network flows to indicate the type of activity or attack present, as shown in Figure 3,4. Here are the class labels available in the dataset:

1. BENIGN: This label represents normal or legitimate network traffic that does not exhibit any malicious behavior.
2. FTP-Patator: This label is associated with attacks using the File Transfer Protocol (FTP) to launch brute-force attacks or gain unauthorized access to FTP servers.
3. SSH-Patator: This label corresponds to attacks that utilize the Secure Shell (SSH) protocol to launch brute-force attacks or unauthorized access attempts on SSH servers.
4. DoS Slowloris: This label indicates a type of Denial-of-Service (DoS) attack known as Slowloris. It involves sending partial HTTP requests to a target server, consuming server resources and preventing legitimate connections.
5. DoS SlowHTTPTest: This label represents the SlowHTTPTest DoS attack, which aims to exhaust server resources by sending HTTP requests slowly or in small chunks.

6. DoS Hulk: This label corresponds to the Hulk DoS attack, where the attacker floods the target server with a high volume of HTTP GET or POST requests, overwhelming its resources.
7. DoS GoldenEye: This label is associated with the GoldenEye DoS attack, which targets web servers by flooding them with slow HTTP POST requests, consuming server resources and causing denial of service.
8. Heartbleed: This label indicates the presence of the Heartbleed vulnerability exploit, a security flaw in OpenSSL that can lead to unauthorized disclosure of sensitive information.
9. Infiltration: This label represents an attack where an unauthorized user gains access to a network or system with malicious intent, typically bypassing security measures.
10. Web Attack – Brute Force: This label signifies a web-based attack involving repeated login attempts using various username and password combinations to gain unauthorized access.
11. Web Attack – XSS: This label corresponds to Cross-Site Scripting (XSS) attacks, where malicious code is injected into web applications, potentially allowing unauthorized access or information theft.
12. Web Attack – SQL Injection: This label indicates the presence of attacks exploiting SQL Injection vulnerabilities, where malicious SQL queries are injected into web applications to gain unauthorized access to databases or manipulate data.
13. Bot: This label represents network traffic associated with botnets, which are networks of compromised computers controlled by a central entity and used for various malicious activities.
14. PortScan: This label indicates the presence of a network scan, where an attacker systematically scans a range of IP addresses to identify open ports and potential vulnerabilities.
15. DDoS: This label corresponds to Distributed Denial-of-Service (DDoS) attacks, where multiple compromised systems are coordinated to flood a target network or server, causing service disruption.

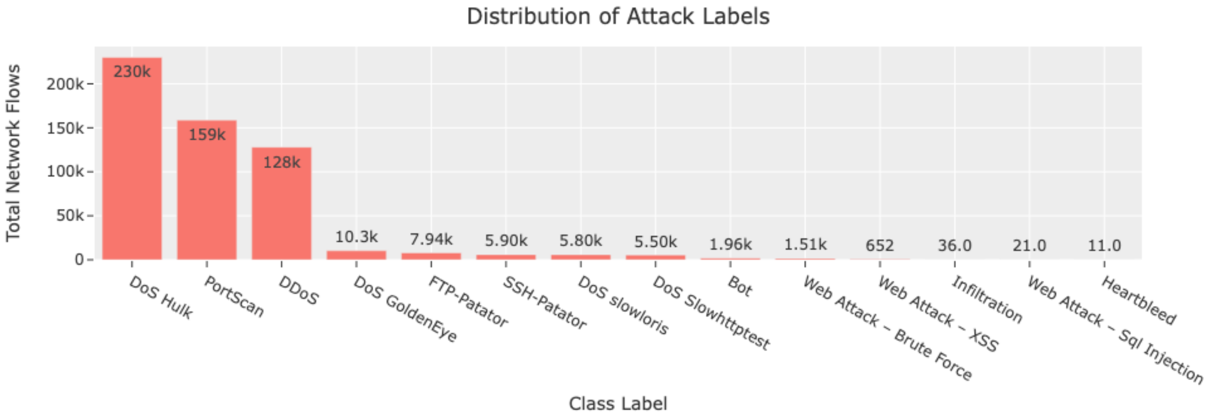


Figure 4: Distribution of Attack Labels in the CIC-IDS2017 Dataset

2.1.2 UNSW-NB15

The UNSW-NB15 dataset is a widely used network intrusion detection dataset developed by the University of New South Wales (UNSW) in Australia [12]. The dataset was created by capturing real network traffic in a controlled environment, simulating both normal and malicious activities. The network traffic is generated using a combination of tools, including the Metasploit framework, to emulate real-world attack techniques and methodologies. The dataset provides over 40 network flow features, which are derived from the network traffic data using Argus and Bro-IDS. The dataset is divided into four CSV files, each representing a specific category of attacks. The dataset also provides access to the raw PCAP files containing the original packet-level data captured during the network traffic collection process.

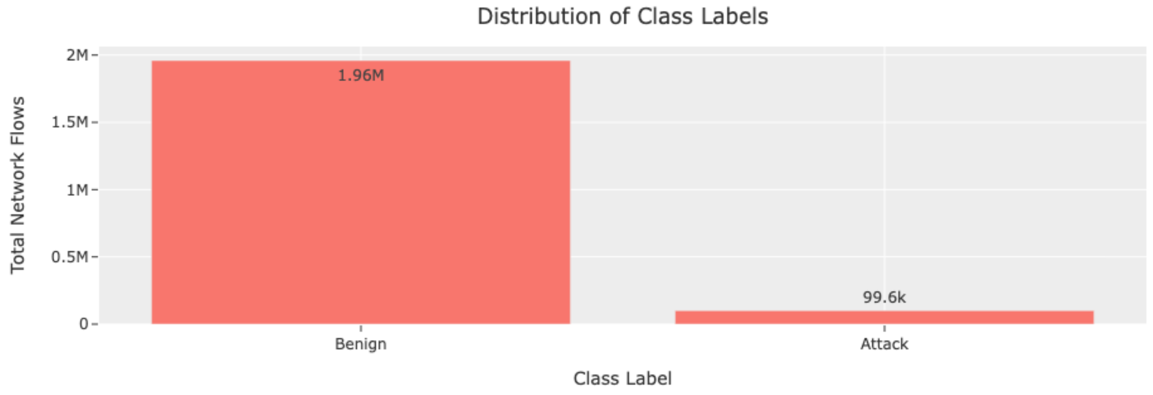


Figure 5: Distribution of Class Labels in the UNSW-NB15 Dataset

The UNSW-NB15 dataset provides 10 class labels that are assigned to network flows to indicate the type of activity or attack present, as shown in Figure 5,6. Here are the class labels available in the dataset:

1. BENIGN: This label represents normal and legitimate network traffic that does not exhibit any malicious or anomalous behavior.
2. Exploits: This label is associated with network traffic instances that exploit software, systems, or protocol vulnerabilities.
3. Reconnaissance: This label corresponds to network traffic instances that involve reconnaissance activities, where an attacker gathers information about a target network or system.
4. DoS: This label indicates network traffic instances associated with denial-of-service attacks that aim to disrupt the availability of a network or system by overwhelming it with excessive traffic or resource consumption.
5. Generic: This label represents instances that may exhibit suspicious or abnormal behavior but cannot be classified into a more specific attack category.

6. Shellcode: This label represents network traffic instances, including shellcodes designed to exploit vulnerabilities and execute arbitrary commands on a target system.
7. Fuzzers: This label corresponds to network traffic instances that use fuzzing techniques to send abnormal or malicious data inputs to test the response and vulnerability of target systems.
8. Worms: This label is associated with network traffic instances representing self-replicating malicious software that spreads across networks by exploiting vulnerabilities to infect and compromise vulnerable systems.
9. Backdoor: This label indicates the presence of network traffic instances related to backdoors that are hidden access points or malicious software that provide unauthorized access to a system, often allowing remote control or unauthorized activity.
10. Analysis: This label represents network traffic instances that involve activities focused on collecting information about a target network or system.

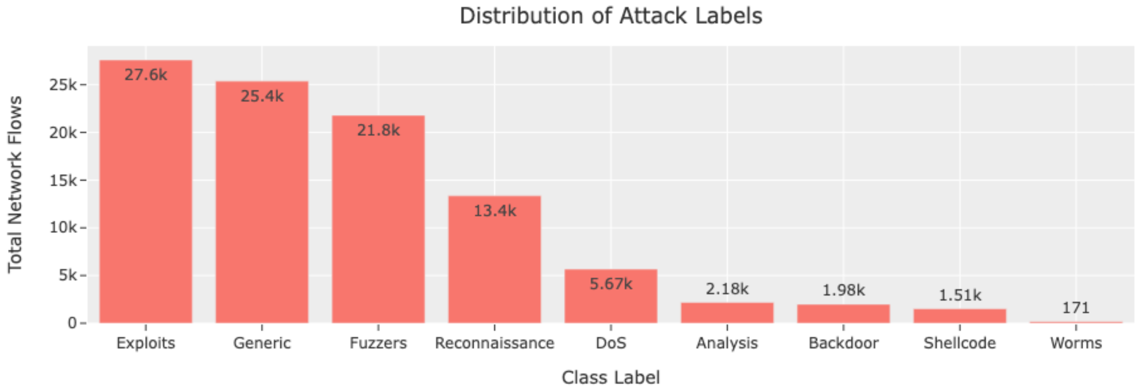


Figure 6: Distribution of Attack Labels in the UNSW-NB15 Dataset

2.2 Data Extraction

In the data processing pipeline, the first step involves extracting packet-level information from the PCAP (Packet Capture) files and assigning appropriate labels. To streamline the data, all CSV files from both the CIC-IDS2017 and UNSW-NB15 datasets are merged into a single CSV file for each dataset, consolidating the relevant data for analysis. Additionally, the raw PCAP files are retrieved from the respective dataset repositories to extract the required packet information. The CIC-IDS2017 dataset comprises five PCAP files for each day of the week, while the UNSW-NB15 dataset contains 80 PCAP files. These PCAP files capture thousands to millions of raw packets, each representing network traffic recorded during the dataset creation process. The Scapy packet manipulation library in Python is used to parse and extract the packet information from the PCAP files [21].

The packet format used in the CIC-IDS2017 PCAP files is Ethernet, which represents the data link layer of the OSI model. On the other hand, the UNSW-NB15 dataset used Cooked Linux as the packet format in the data link layer. The Scapy library is utilized to parse each packet within the PCAP files, enabling the extraction of the network layer and transport layer protocols, as well as the specific fields within them. The extracted protocols, header fields, and their corresponding values are recorded in CSV files. Additionally, the entire packet and payload contents are added to the CSV file, represented in hexadecimal format. The labeling process is crucial to assign the correct attack labels to each packet, aligning them with the relevant timestamp and matching columns in the original CSV files that contain the flow information. Following the methodology outlined in the Payload-Byte paper [22], the packets are appropriately labeled based on the attack type. Finally, 10 labeled CSV files for the CIC-IDS2017 dataset and 17 labeled CSV files for the UNSW-NB15 dataset are created. Each file contains a comprehensive collection of 10 million packets, accompanied by their header information in both the network and transport layers, and payload.

2.3 Data Integration

The process of data integration follows the previous steps and involves creating three distinct subsets of datasets from each CSV file in both the UNSW-NB15 and CIC-IDS2017 datasets. These subsets are categorized as follows:

1. Packet Fields
2. Packet Bytes
3. Payload Bytes

The Packet Fields subset encompasses all the header fields of the network packet. It includes comprehensive information about the packet, such as source and destination IP addresses, TCP ports, TTL, Sequence numbers, Flags, Options, packet length, and various other relevant fields. Additionally, the subset incorporates the entire packet and payload in hexadecimal format, allowing for a detailed analysis of the packet structure when needed. The Packet Bytes subset captures the complete byte representation of the network packet, including the payload. However, in this subset, the byte values are converted from hexadecimal to decimal format, ranging from 0 to 255. This conversion facilitates further data processing and analysis, enabling researchers to apply numerical techniques and algorithms to the byte-level data. The Payload Bytes subset focuses specifically on the byte representation of the payload within the network packet. Similar to the Packet Bytes subset, the payload bytes are converted from hexadecimal to decimal format, providing a more interpretable representation of the payload data. This subset is particularly useful when analyzing the content of specific characteristics of the payload, such as detecting patterns or identifying anomalies. To ensure usability and accessibility, the three subsets of the datasets are processed and cleaned, resulting in the creation of processed CSV files. Each CSV file corresponds to a specific subset and dataset combination, resulting in a total of 108 CSV files (18 CSV files for each subset from both UNSW-NB15 and CIC-IDS2017 datasets). These CSV files serve as valuable resources for further research and analysis by other researchers and practitioners in the field of network security and intrusion detection.

In addition to the packet-level datasets, we have also integrated flow information to enhance the usability and analysis of the dataset. By combining the flow information with the packet-level data, researchers can gain a more comprehensive understanding of the network traffic and make informed observations and inferences. To achieve this integration, we processed and cleaned the original flow datasets of both UNSW-NB15 and CIC-IDS2017 and named them "Network Flows". These flow datasets contain information about the network flow, including source and destination IP addresses, ports, protocols, aggregated information like bytes per second, packets per second, flow duration, and other relevant attributes. We then mapped the flow information from these datasets to each of the 108 CSV files generated in the previous steps. The mapping process involved matching the source and destination IP addresses, ports, protocols, and class labels between the flow datasets and the corresponding packet-level CSV files. This matching allowed us to establish a relationship between the packet-level data and the flow information, linking the detailed packet-level insights with the broader flow context. These flow datasets include the flow information, along with a unique "flow-id" assigned to each flow. The flow labels were also retained, providing a valuable reference for understanding the nature of the flows. In each of the 108 packet-level datasets, a new column called "flow_id" was added. This column served as the key to match and join the packet-level data with the corresponding flow information from the Network Flows dataset. Furthermore, each CSV file contains a "packet_id" column, which serves as a unique identifier for each packet within the dataset.

We have developed a dedicated Python package (nids-datasets) to facilitate the seamless acquisition and utilization of the dataset. This package offers convenient functionalities that enable users to download the necessary files directly into their working code environment. By incorporating this package into their workflow, researchers and practitioners can easily access the dataset without the need for manual file handling. The package also includes features that simplify the process of joining the flow dataset with the packet-level dataset. In the "Packet Bytes" and "Payload Bytes" subsets, the corresponding CSV files contain the first 1600 and 1500 bytes, respectively, of each packet. The package provides functionality for generating the decimal representation of bytes for the entire packet and payload, up to the maximum of 65535 bytes.

Table 1 presents detailed statistics of the curated dataset, providing information on the number of packets in each CSV file and the frequencies of different labels within those files. These statistics offer valuable insights for researchers, enabling them to selectively download and utilize the essential files using our Python package, rather than acquiring the entire dataset. Our Python package streamlines this process by integrating with the Hugging Face Datasets library, to seamlessly download and use the specific files with zero-copy reads and efficient memory management. The CSV files are converted to Apache Parquet files for optimized columnar storage and efficient data retrieval.

Table 1: Total Packets and Labels on our curated Dataset CSV Files

CSV File	**UNSW-NB15** (Total Packets and labels)	**CIC-IDS2017** (Total Packets and labels)
1	9990118 normal:9597932, exploits:219231, dos:66031, fuzzers:61609, generic:28211, reconnaissance:12572, worms:1732, shellcode:1459, backdoor:953, analysis:388	4822905 BENIGN:4822905
2	9971265 normal:9771699, exploits:121153, dos:33817, generic:17799, fuzzers:17397, reconnaissance:6820, analysis:1086, shellcode:598, backdoor:572, worms:324	3571097 BENIGN:3571097
3	9980200 normal:9980200	3558479 BENIGN:3556526, FTP-Patator:1953
4	9979823 normal:9979823	2588707 BENIGN:2575904, FTP-Patator:12803
5	9982881 normal:9982881	3075291 BENIGN:3007086, FTP-Patator:68205
6	9983244 normal:9983244	2705079 BENIGN:2637797, SSH-Patator:38680, FTP-Patator:28602
7	9984416 normal:9984416	2737246 BENIGN:2612615, SSH-Patator:124631
8	9986659 normal:9986659	2649076 BENIGN:2649076
9	9987598 normal:9987598	3015813 BENIGN:2948428, DoS slowloris:47519, DoS Slowhttptest:19866

(cont. on next page)

Table 1 (cont.)

CSV File	**UNSW-NB15** (Total Packets and labels)	**CIC-IDS2017** (Total Packets and labels)
10	9996833 normal:9672644, exploits:208672, dos:46811, fuzzers:26704, generic:25510, reconnaissance:12166, backdoor:1418, shellcode:1262 worms:1168, analysis:478	1901467 DoS Hulk:1459567, BENIGN:422225, DoS Slowhttptest:19675
11	9997283 normal:9625429, exploits:216055, dos:69137, fuzzers:38687, generic:29047, reconnaissance:13541, analysis:1520, shellcode:1306, worms:1298, backdoor:1263	5359118 BENIGN:4418585, DoS Hulk:786683, DoS GoldenEye:104513, Heartbleed:49296, DoS slowloris:41
12	9995911 normal:9583173, exploits:231463, dos:71735, fuzzers:42034, generic:36914, reconnaissance:24232, analysis:2024, backdoor:1688, shellcode:1434, worms:1214	2663758 BENIGN:2651388, Web Attack – Brute Force:12370
13	9995903 normal:9618777, exploits:226910, dos:57220, fuzzers:44183, generic:29415, reconnaissance:14069, backdoor:1809, worms:1688, shellcode:1396, analysis:436	3136779 BENIGN:3109436, Web Attack – Brute Force:17873, Web Attack – XSS:9344, Web Attack – SQL Injection:126
14	9994804 normal:9636038, exploits:199265, dos:72111, fuzzers:40441, generic:28263, reconnaissance:13001, analysis:1784, backdoor:1427, shellcode:1272, worms:1202	2905004 BENIGN:2845250, Infiltration:59754
15	9995988 normal:9643297, exploits:222618, dos:44221, generic:33660, fuzzers:32462, reconnaissance:13400, analysis:2224, backdoor:1928, shellcode:1300, worms:878	2558990 BENIGN:2558990
16	9996434 normal:9623646, exploits:216629, dos:70959, fuzzers:38901, generic:28577, reconnaissance:12487, worms:1780, backdoor:1473, shellcode:1332, analysis:650	3195638 BENIGN:3182740, Bot:12853, PortScan:45
17	9996040 normal:9584978, exploits:231583, dos:73005, fuzzers:44524, generic:41717, reconnaissance:13645, analysis:2009, backdoor:1821, worms:1490, shellcode:1268	3184487 BENIGN:1920222, DDoS:942879, PortScan:321386
18	9755809 normal:9403900, exploits:217538, dos:46603, fuzzers:34134, generic:33750, reconnaissance:14262, backdoor:1932, analysis:1476, shellcode:1356, worms:858	817864 BENIGN:528653, DDoS:289211

2.4 Data Transformation

We performed data transformation on the curated dataset to ensure the data were suitable for further analysis and aligned with our specific research objectives. Several steps were taken to filter and manipulate the data to create a dataset that meets our requirements. Firstly, we applied a filtering process to include only TCP/IP packets in our research focused on generating tags and descriptions. The header fields present in the TCP/IP are shown in Figure 7. Next, we selected packets that contain a payload, excluding packets that serve the purpose of connection establishment. While important for network communication, these connection control packets do not typically carry application-level data and might not contribute to our specific analysis goals. To address memory and computational constraints, we sampled a maximum of 50,000 packets per label from both the UNSW-NB15 and CIC-IDS2017 datasets.

Using the 18 CSV files from each dataset subset, we collected the samples and joined the records from the "Packet Fields" and "Payload Bytes" subsets using the matching "packet_id" column. This merge process consolidates the packet header fields and the numerical representation of the payload bytes, creating a comprehensive dataset that encompasses both components. Furthermore, we incorporated the flow information dataset by joining it with the transformed packet-level dataset. The matching "flow_id" column served as the key to combining the flow information. As a result of these data transformations and integrations, the final datasets were generated, each containing individual packets along with their corresponding header information, payload bytes, and flow information.

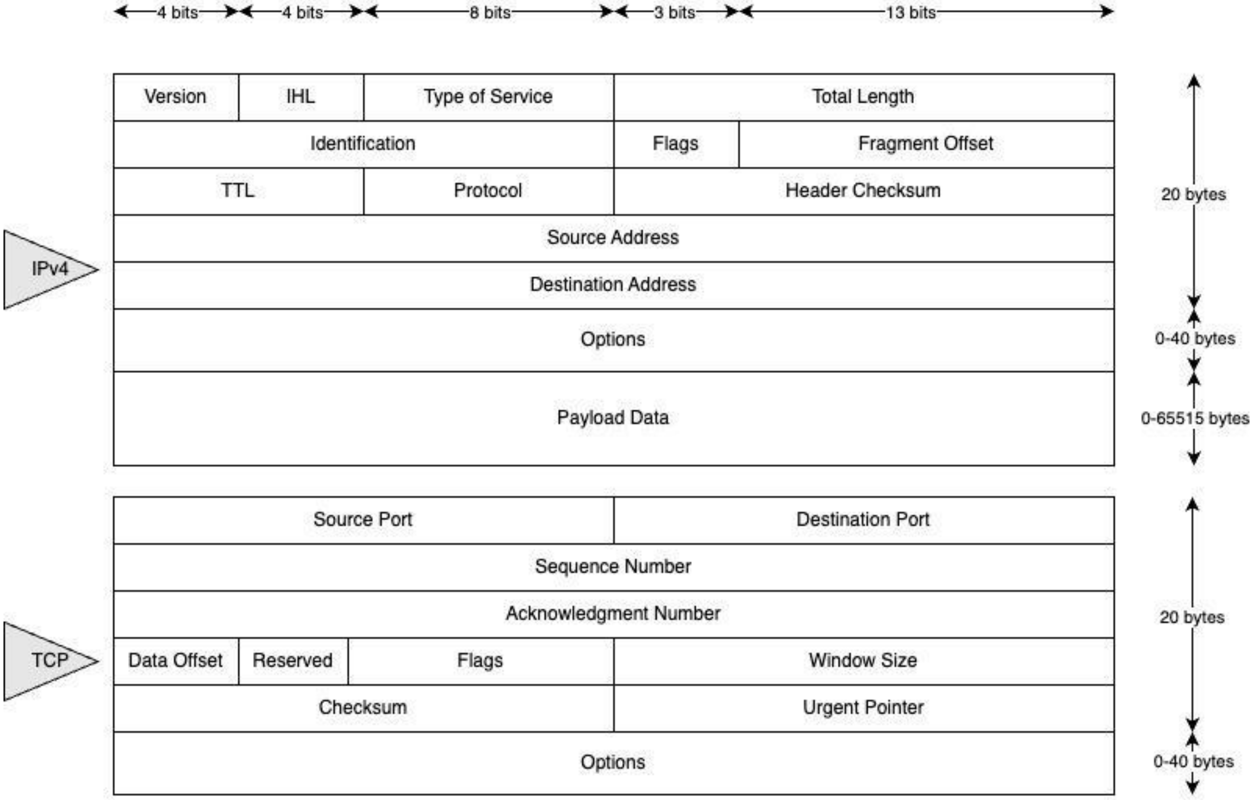


Figure 7: TCP/IP Packet Structure

Considering that both the UNSW-NB15 and CIC-IDS2017 datasets are created within a closed environment, we decided to remove certain header information from the dataset. This included eliminating source and destination IP addresses, which are specific to the closed environment and may not contribute significantly to our research objectives. Furthermore, we eliminated other less critical header information, such as TCP sequence and acknowledgment numbers, TCP options, TCP checksum, and frame information. While these details are crucial for specific network operations, they may not directly impact our research goals related to intrusion detection and network traffic analysis.

To enhance the dataset and capture additional information, we introduced a new column called "payload_length." This column enables us to calculate and store the length of the payload for each packet. To handle categorical features within the dataset, such as IP ToS (Type of Service) and TCP flags, we performed label encoding. In addition to the packet-level data, we extracted and included three new columns from the existing flow information. These columns capture the number of forward packets per second, backward packets per second, and bytes transferred per second. This information provides valuable insights into the flow rate and volume, allowing for more efficient intrusion detection and real-time analysis.

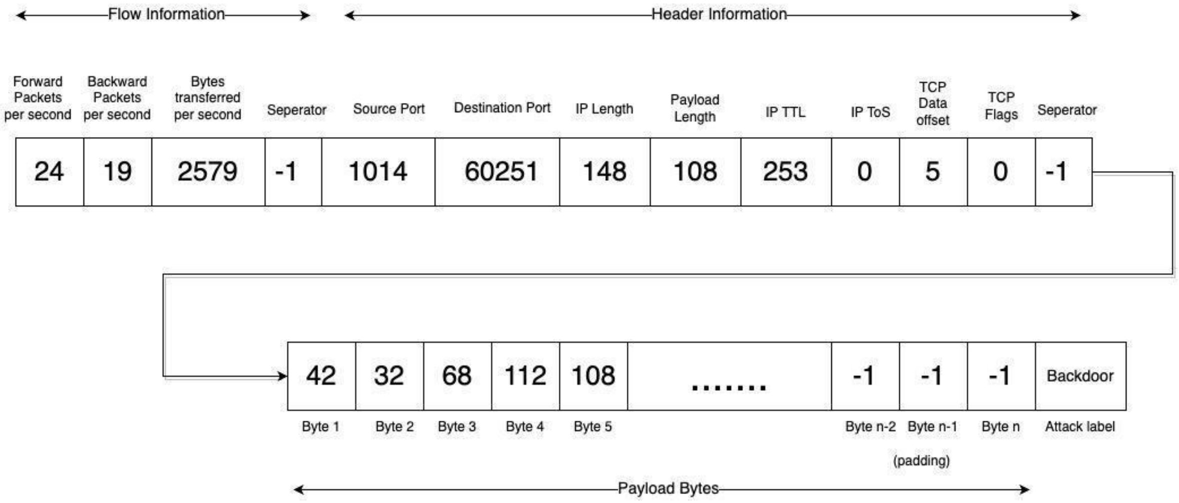


Figure 8: Data Format for Tag Generation

We removed all other flow-related details to optimize the dataset and ensure that the remaining information is relevant and useful for the analysis and modeling tasks. We have also introduced two separator columns with the constant value of -1 that separates the flow information columns, header information columns, and payload bytes columns. Figure 8 depicts the final structure of the transformed dataset, which is specifically designed for the tag generation task.

For the text generation task, we have created a dictionary that contains all the header fields along with their respective values for each packet. This dictionary is then converted into a string representation, preserving the structure and information of the header fields. Regarding the payload, we have converted it to a readable ASCII format, ensuring that the content is in a human-readable form. If the payload contains any English words, we have extracted those words from the payload. These extracted words are then added to the end of the header information string, providing additional context and information for the text generation task.

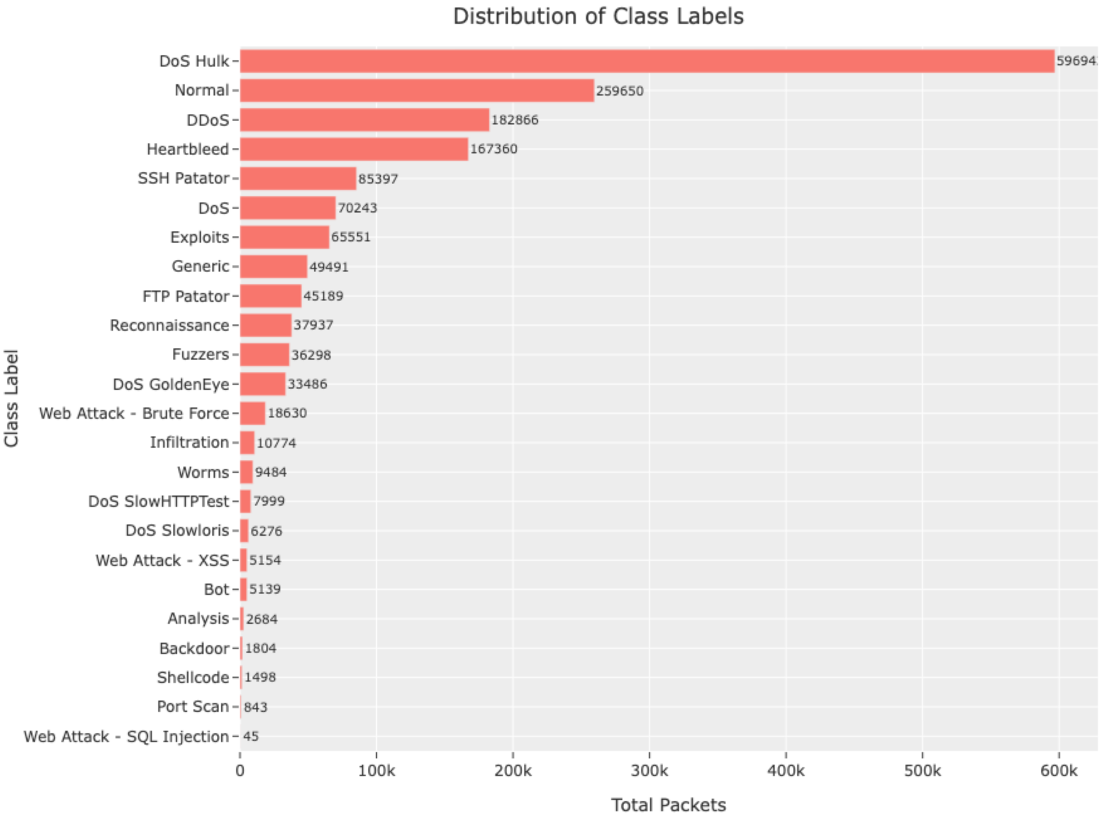
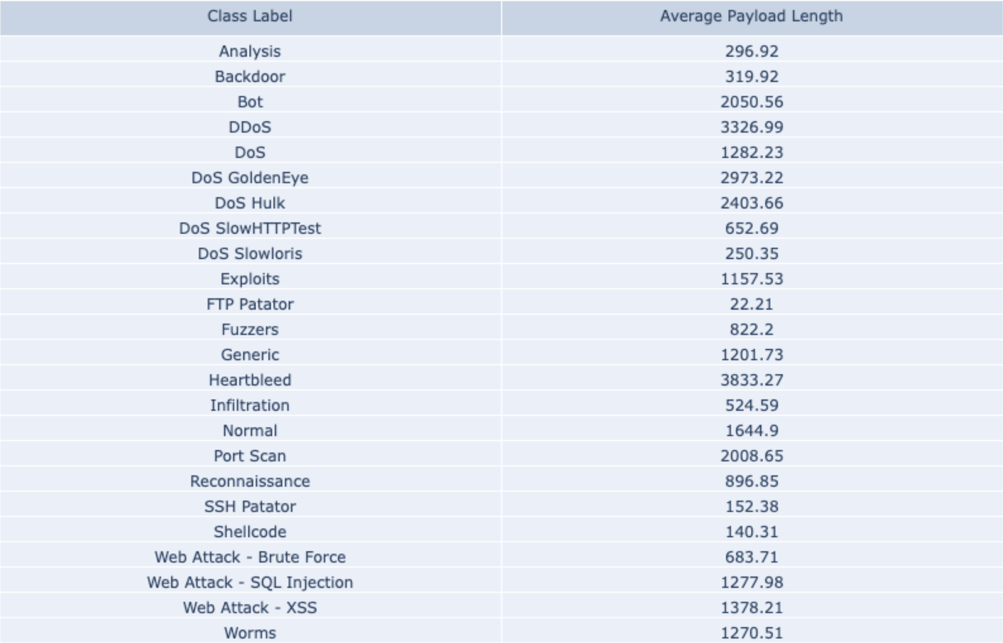


Figure 9: Distribution of Class Labels in the Final Combined Dataset

For improved generalization and convenience in handling the data, we merged the data created from both the UNSW-NB15 and CIC-IDS2017 datasets to create a unified combined dataset. In this process, we employed a strategy to combine the normal or benign packets from each dataset into a single label, while leaving the other labels unchanged since they represent distinct types of network activities. To ensure data integrity and avoid redundancy, duplicate records were identified and subsequently dropped from the combined dataset. Figure 9 illustrates the class distribution of the resulting combined dataset.



Class Label	Average Payload Length
Analysis	296.92
Backdoor	319.92
Bot	2050.56
DDoS	3326.99
DoS	1282.23
DoS GoldenEye	2973.22
DoS Hulk	2403.66
DoS SlowHTTPTest	652.69
DoS Slowloris	250.35
Exploits	1157.53
FTP Patator	22.21
Fuzzers	822.2
Generic	1201.73
Heartbleed	3833.27
Infiltration	524.59
Normal	1644.9
Port Scan	2008.65
Reconnaissance	896.85
SSH Patator	152.38
Shellcode	140.31
Web Attack - Brute Force	683.71
Web Attack - SQL Injection	1277.98
Web Attack - XSS	1378.21
Worms	1270.51

Figure 10: Mean Payload Length for each Class Label in Bytes

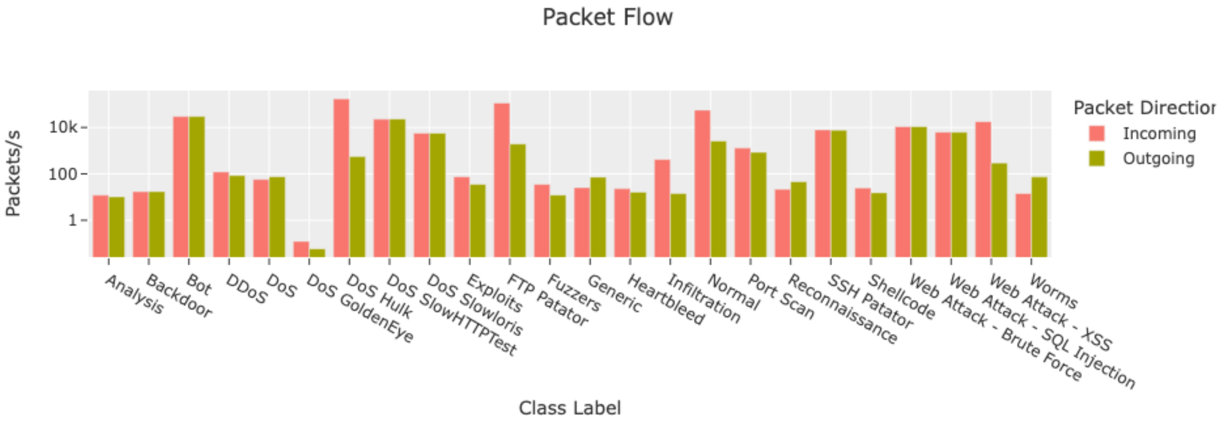


Figure 11: Packet Flows per second for each Class Label

In Figure 10, the analysis reveals the average payload length for each class label. The findings indicate that FTP Patator attacks exhibit a relatively low mean payload length of 22 bytes. On the other hand, Heartbleed attacks exhibit the highest mean payload length among the observed attack types. Shifting the focus to the flow perspective, Figure 11 presents the total number of packet flows per second for each class label. It becomes evident that the DoS Hulk attack demonstrates a high packet flow rate of more than 10,000 packets per second. In contrast, the DoS Golden Eye attack exhibits an extremely low packet flow rate, with less than one packet per second. This emphasizes the disparity in the intensity and impact between these two types of denial-of-service attacks.

Chapter 3: Methodology

3.1 Overview

Figure 12 illustrates the subsequent steps in the process, following the preparation of the combined dataset. In this phase, the dataset is utilized to fine-tune the BERT model [19] and generate embeddings to understand the specific characteristics and semantics of network packets.

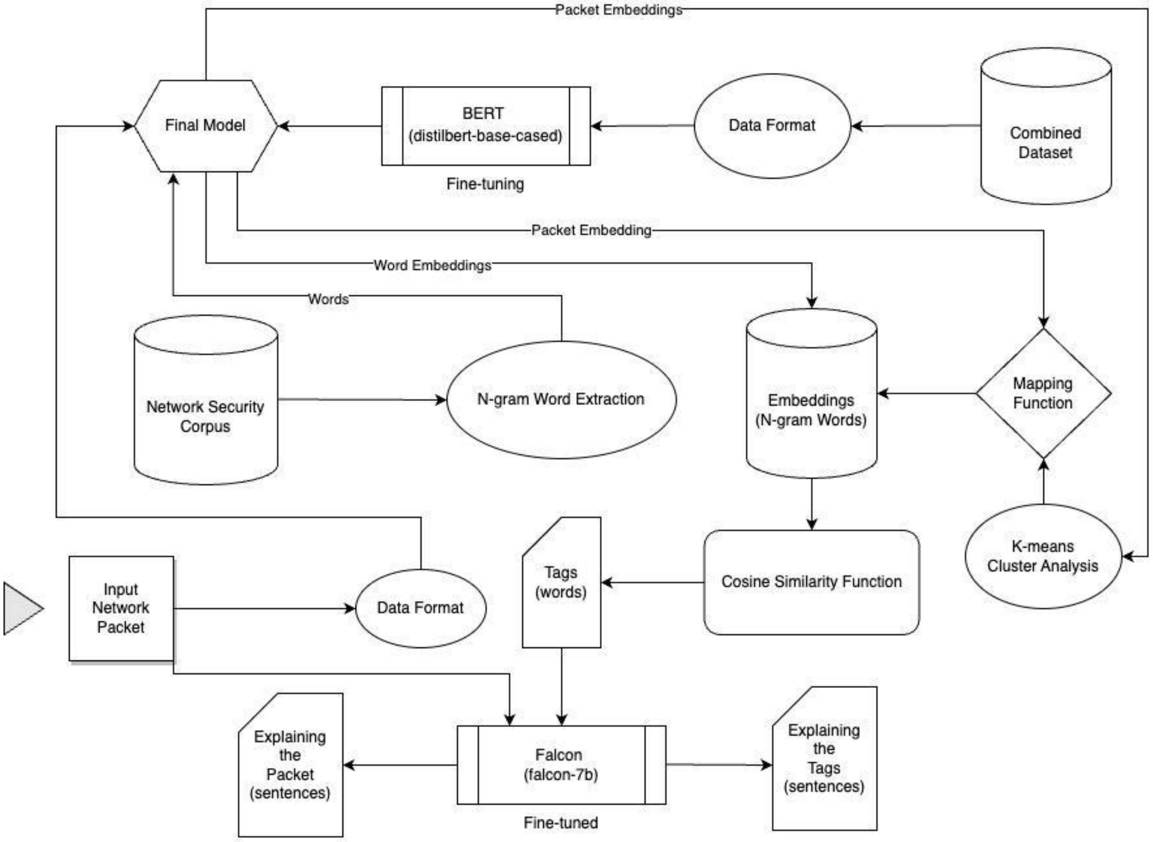


Figure 12: High-level Overview of Tag and Text Generation Processes

3.2 Word Embeddings

In our research, embeddings play a crucial role as the foundational elements. Embeddings are numerical representations of words or sentences that enable machine learning models to comprehend and process textual data. Word embeddings are specifically designed to represent individual words, while sentence embeddings capture the overall meaning of a sequence of words, representing the entire sentence. There are various methods available for generating word embeddings, two of the most well-known being Word2vec [23] and GloVe [24]. These techniques leverage models trained on Natural Language Processing (NLP) tasks to derive meaningful and dense vector representations for words. Word2Vec models are trained to capture semantic relationships and contextual information by learning word embeddings based on the surrounding words in a large corpus of text. Similarly, GloVe (Global Vectors for Word Representation) creates word embeddings by analyzing word co-occurrence statistics across a corpus. These embeddings capture word similarities and contextual meaning, allowing the model to understand the semantic relationships between words. In addition to Word2vec and GloVe, transformer models such as BERT (Bidirectional Encoder Representations from Transformers) have gained significant popularity in the field of NLP for generating embeddings. BERT employs a transformer architecture that considers the context from both the left and right sides of a word, capturing a deeper understanding of word meaning and sentence context. BERT embeddings have proven to be highly effective for various NLP tasks, including sentence classification, named entity recognition, and sentiment analysis.

3.2.1 Word2vec

Word2Vec is a powerful framework that generates word embeddings by using a neural network, which captures the context information of words that appear in similar contexts and tend to have similar meanings. The framework provides two approaches for training word embeddings: Continuous Bag-of-Words (CBOW) and Skip-gram [25].

The CBOW approach in Word2Vec focuses on predicting the target word based on its surrounding context words. The model learns to generate word embeddings by maximizing the probability of correctly predicting the target word given the context words. On the other hand, the Skip-gram approach aims to predict the surrounding context words given a specific target word. By learning to predict the context words, the model generates word embeddings that encapsulate the contextual information of the target word. Word2vec models use a shallow neural network architecture with a single hidden layer to learn the word embeddings. Through the training process on a large corpus of text data, the model adjusts the embeddings in a high-dimensional space. To study the word embeddings generated by the Word2vec model, the Gensim library is commonly used. Each word in the Word2vec model has a unique embedding that represents its meaning and context. By examining the vector space, we can observe that similar words tend to cluster together, indicating the model's ability to capture semantic relationships. Cosine similarity is a commonly used metric to measure the similarity between word embeddings. By calculating the cosine similarity between two embeddings, we can identify the nearest embeddings which represent words similar to the target word. This similarity measurement allows for tasks such as finding synonyms, identifying related terms, or even detecting analogies within the word vector space.

To visualize the word embeddings, we have used the trained Word2vec model to retrieve the embeddings for all the words present in our five sentences. These embeddings have a dimension length of 300 and are the numerical representations that capture the word's semantic and contextual meaning. We employed the t-SNE (t-Distributed Stochastic Neighbor Embedding) algorithm [26], which is a dimensionality reduction technique that reduces the embeddings to two dimensions while preserving the relative distances between the data points.

The five sentences used for visualization are as follows:

Sentence 1: "The scientist analyzed the effects of lead exposure on the human body."
Sentence 2. "The engineer used a lead screw to precisely control the movement of the robotic arm."
Sentence 3. "The talented actor played the lead role in the critically acclaimed movie."
Sentence 4. "The athlete sprinted ahead, taking an early lead in the race."
Sentence 5. "The inspirational leader guided the team to success with their vision and expertise."

Each of these sentences demonstrates a distinct context in which the word "lead" is used. The first sentence refers to the toxic metal, "lead," and its effects on the human body. The second sentence describes the use of a mechanical component, a "lead screw," in robotic arm control. The third sentence highlights the "lead" role played by an actor in a movie. The fourth sentence pertains to an athlete taking an "early lead" in a race. Lastly, the fifth sentence portrays a visionary and influential "leader" guiding a team to success.

Figure 13 depicts the word embeddings generated by Word2vec for all the words present in the five given sentences. It provides a visual representation of the embedding space, highlighting the positioning of words and their semantic relationships. The word "lead" appears only once in the visualization and has a single corresponding embedding. We can also observe that similar words tend to cluster together in the embedding space. This clustering indicates that words with similar meanings or contextual associations are positioned near one another. This property of word embeddings allows for capturing semantic relationships and contextual similarities between words.

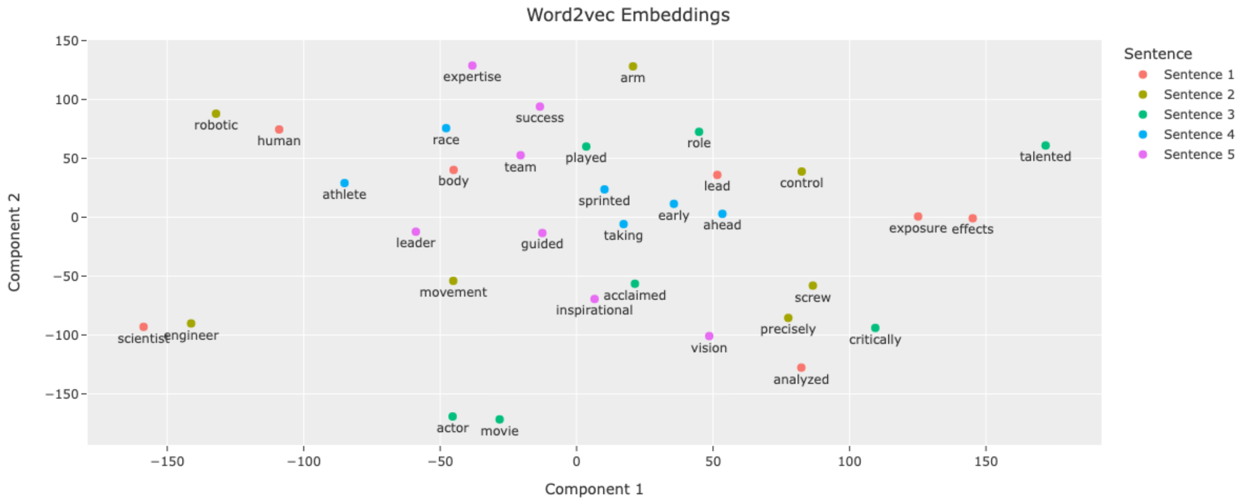


Figure 13: Word2vec Embeddings Visualization

3.2.2 GloVe

GloVe (Global Vectors for Word Representation) is another method for generating word embeddings based on the co-occurrence statistics of words in a corpus [24]. It combines the advantages of global matrix factorization methods with the local context window-based approaches used by Word2vec. GloVe constructs a word-to-word co-occurrence matrix, representing the frequency of words appearing together in a given context window. This matrix is then factorized using matrix factorization techniques to generate word embeddings. GloVe can capture word similarities and relationships across the entire corpus rather than relying on local context windows. Similar to Word2vec, GloVe-generated word embeddings can measure semantic similarity, find nearest neighbors, or serve as input representations for various NLP tasks. Figure 14 depicts the word embeddings generated by GloVe for all the words present in the five given sentences. It also provides a visual representation of the embedding space similar to Word2vec, highlighting the positioning of words and their semantic relationships.

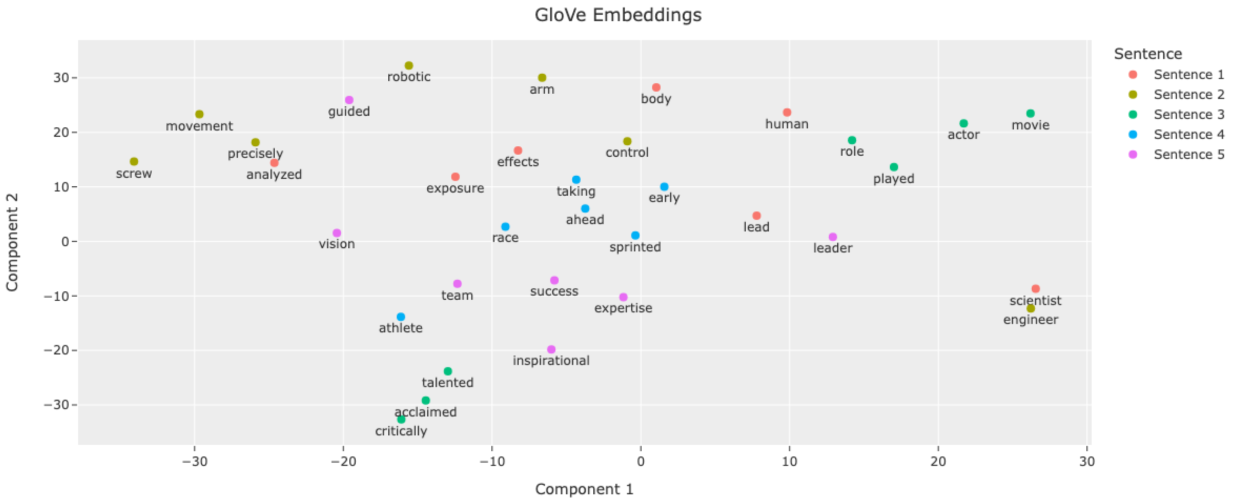


Figure 14: GloVe Embeddings Visualization

3.2.3 BERT

BERT is a transformer-based model that considers the bidirectional context of words, capturing both the preceding and following words to understand their meaning within a sentence [19]. This enables BERT to generate contextualized word embeddings that are highly effective in capturing intricate semantic relationships. Unlike Word2vec and GloVe, which generate static word embeddings based on co-occurrence local context, BERT takes advantage of its transformer architecture to create dynamic word embeddings. BERT is pre-trained on a large corpus of text, using masked language modeling and next-sentence prediction tasks. This pre-training enables BERT to capture rich contextual information and produce word embeddings that adapt to the specific context in which the word appears.

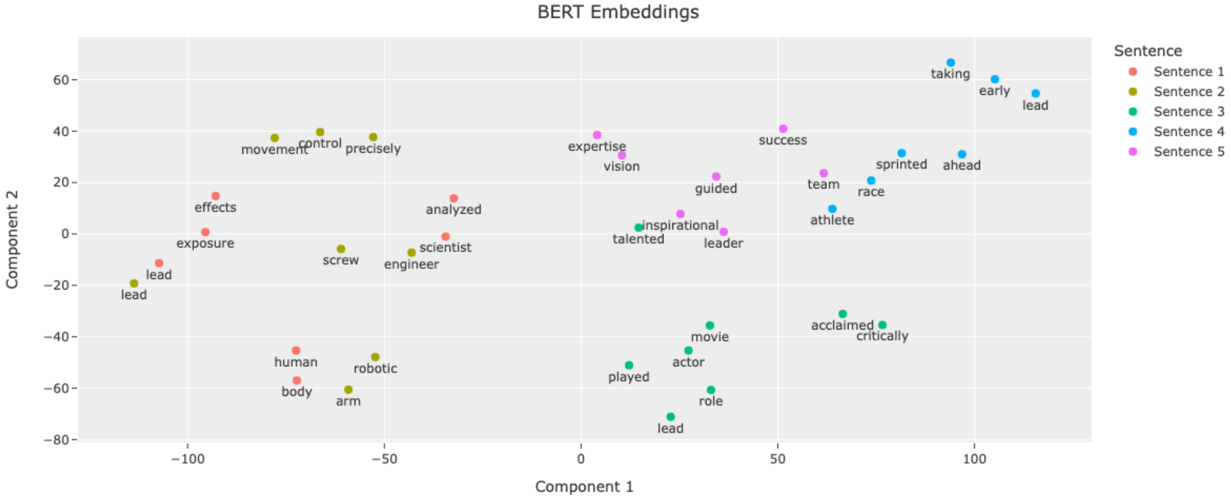


Figure 15: BERT Embeddings Visualization

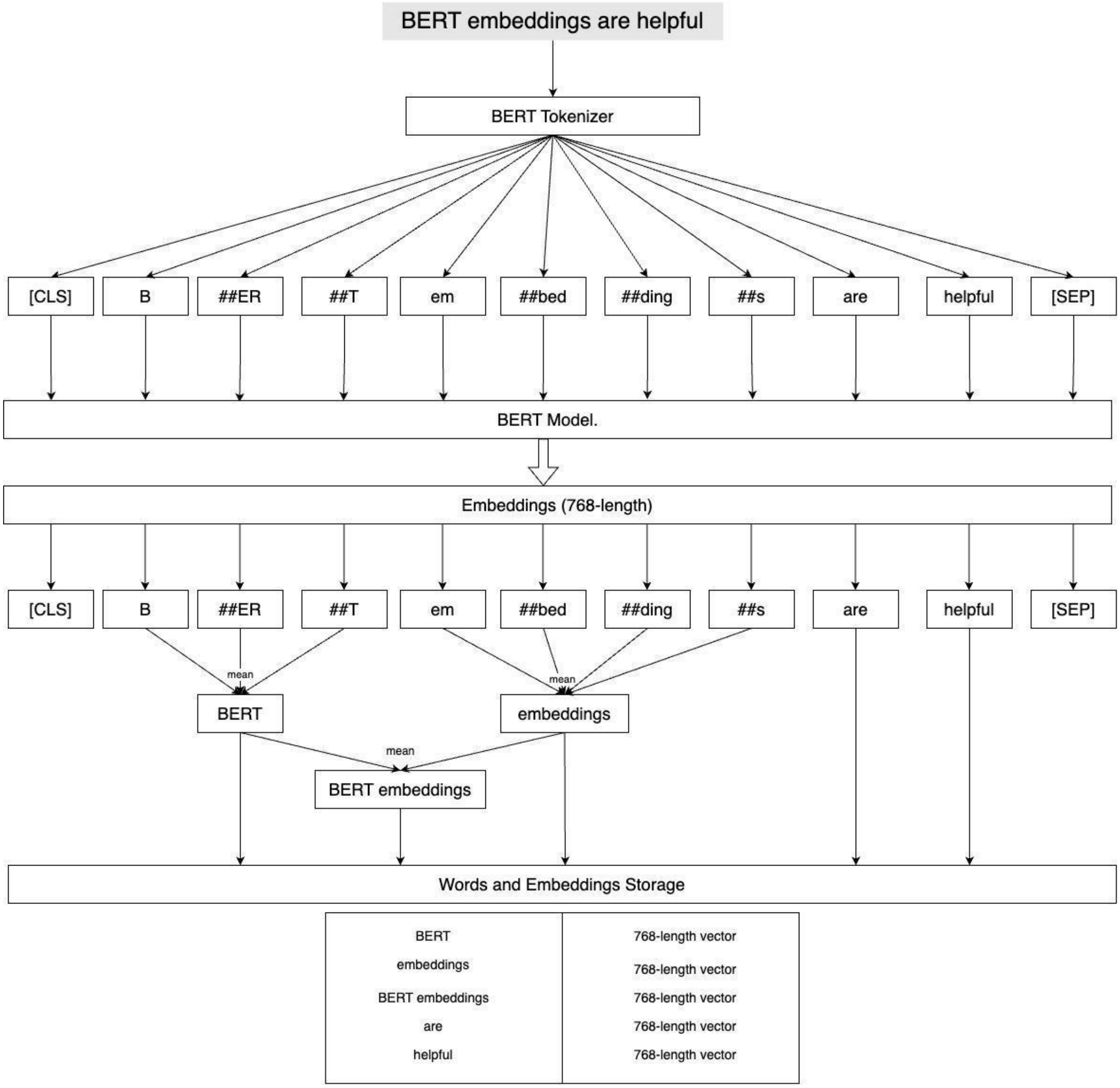
BERT generates word embeddings of vector length 768, where each embedding captures the contextualized meaning of words based on the sentence, they appear in. In Figure 15, we visualize the embeddings of words present in the five given sentences using t-SNE. Notably, unlike Word2Vec and GloVe, the words in each sentence form distinct and well-formed clusters based on the sentence they belong to. This clustering showcases the power of BERT in capturing contextual information and understanding the semantics specific to each sentence. The word "lead," present in all five sentences, has different embeddings based on the contextual information of each sentence. This demonstrates the contextualized nature of BERT embeddings, where the same word can have different representations depending on its context. Within each cluster of sentences, the words that share semantic similarity are positioned close together. These properties of BERT embeddings, including their contextualized nature and ability to capture rich semantic relationships, make them well-suited for various natural language processing tasks. In the context of the paper, these characteristics are leveraged to generate tags for the packets. By utilizing the embeddings, the model can identify similarities and relationships between packets, enabling the generation of meaningful tags for the packets, even for those falling into new categories.

3.3 BERTSimilar

Creating word embeddings using Word2Vec and GloVe models through the Gensim Python library [27] is a straightforward process. These models generate static embeddings, meaning that each word has a fixed and unique embedding representation. These embeddings can be saved in a file and utilized in various tasks since they remain the same across different contexts. On the other hand, generating and storing embeddings using BERT poses challenges due to the dynamic nature of BERT embeddings. BERT produces contextualized embeddings that vary for the same word based on its surrounding context. This dynamic nature makes it difficult to directly generate and store BERT embeddings for future use. To address this challenge and leverage BERT embeddings in our project, we have developed an advanced Python package called BERTSimilar. BERTSimilar facilitates the generation of BERT embeddings and supports subsequent tasks such as retrieving similar words related to a given input.

BERT embeddings can be generated for the words in a given dataset using the SimilarWords module. This module accepts various types of input, including Wikipedia pages, word documents, and text files, and proceeds to generate embeddings for all the words present in the extracted dataset. To accommodate the maximum sequence length supported by the BERT model, the module initially splits the dataset into paragraphs consisting of 512 words. Hugging Face Transformers library [31] is used to load the BERT model and the BERT tokenizer, enabling efficient tokenization and embedding generation. Each paragraph is tokenized using the BERT tokenizer, resulting in a tokenized output that captures the subword or token representation of the words. This tokenized output is then fed into the BERT model to obtain the embeddings.

The BERT model provides embeddings for each word in the last layer, as well as a pooled output embedding that summarizes the entire paragraph. Since BERT uses a WordPiece tokenizer, words that are not present in the BERT vocabulary are split into subwords or tokens. This ensures that a comprehensive representation is captured for words that may have variations or are out-of-vocabulary. Punctuation marks and stop words are excluded from the tokenized output to focus on meaningful words for the embedding generation. The module collects the embeddings for each word and merges subwords to create the whole word. The embeddings are averaged for joined words that consist of multiple subwords or tokens to obtain a representative embedding for the merged word. This consolidation enabled the module to provide coherent embeddings for both individual and merged words, enhancing the comprehensiveness of the generated embeddings. Furthermore, the SimilarWords module goes beyond single words and also calculates embeddings for bigram, trigram, and other n-gram words present in the input dataset. This allows for a broader scope of embeddings, considering combinations of words and capturing their contextual relationships. The embeddings for these n-gram words are stored alongside the corresponding word representation, helping subsequent analysis and exploration. Figure 16 illustrates how tokenization and embedding generation are handled by the BERTSimilar package.



BERT	768-length vector
embeddings	768-length vector
BERT embeddings	768-length vector
are	768-length vector
helpful	768-length vector

Figure 16: Overview of BERTSimilar Embedding Generation

The SimilarWords module has a find_similar_words() method that is used to get similar words to the given words or embedding. This method takes several parameters including input words, input context, output n-gram, and maximum output words. If the context is given, the embeddings for the input words are retrieved from the paragraphs that are similar to the given context. Next, the method calculates the cosine similarity score between the average embedding of the input words and the embeddings of all other words stored within the module to find similar words. The cosine similarity score measures the similarity between two vectors by calculating the cosine of the angle between them. Based on the computed cosine similarity scores, the method identifies the n-gram words with the highest similarity scores. The closest words and their similarity score are the output of this method. For generating tags for the network packets, a text corpus containing words and paragraphs related to network packets and attacks is utilized. The SimilarWords module is used to create embeddings for all the words within the corpus. By obtaining embeddings for the words in the text corpus, a comprehensive representation of the network packet-related vocabulary is captured. For a given packet as input, the module uses the embeddings to identify the closest words to the embedding of the packet. These closest words, determined by their proximity in the embedding space, are output as tags, effectively describing the characteristics and nature of the packet.

3.4 Model Fine-tuning

The subsequent step in the process involves fine-tuning the BERT model to adapt it to our specific task requirements. Fine-tuning is a process where a pre-trained model, such as BERT, is further trained on a specific task or domain [28] [29]. BERT, like other pre-trained language models, is initially trained on large amounts of text data. This training enables the model to acquire a general understanding of the structure, grammar, and semantics of the language. However, since BERT is not specifically trained on network packet content, it needs to learn and understand packet and payload contents and their relationship. Fine-tuning involves training the pre-trained BERT model using our curated and transformed dataset, which contains packet-level information, payload bytes, and flow details.

To prepare the data for fine-tuning the BERT model, we adopt a specific approach to ensure compatibility with the model's tokenizer, which operates on tokenized inputs. To achieve this, we concatenate all the columns in the dataset, separating them with a space, into a single string. By doing so, each individual feature or column can be treated as a separate token by the WordPiece tokenizer. For the fine-tuning process, we utilize the "distilbert-base-cased" model [30]. This model is a distilled version of BERT, designed to be smaller and faster while maintaining a high level of performance. The "distilbert-base-cased" model offers a 60% improvement in processing speed compared to the base BERT model while preserving approximately 95% of its performance.

BERT has a limitation in terms of its maximum sequence length, which is set at 512 tokens. This restriction means that the BERT model cannot process inputs longer than 512 tokens. While other models like Longformer [32] and BigBird [33] can handle longer sequences, they come with significant computational costs due to the $O(n^2)$ time and memory complexity, where n represents the sequence length. In our analysis, we have determined that the payload bytes at the beginning of the packet are more important in understanding the signature of the payload and its relationship with various network attacks. This finding has practical implications for working with the BERT model, as we can focus on a subset of the payload bytes to achieve efficient and effective fine-tuning. So, we have limited the input to the first 500 payload bytes for fine-tuning the BERT model, where the remaining length is used for flow and header features. Figure 17 provides further insights into the importance of payload bytes. We employ the Random Forest classifier algorithm and examine the feature importance it assigns to the payload bytes. This analysis allows us to measure the relative significance of different payload bytes contributing to the classification or identification of network attacks.

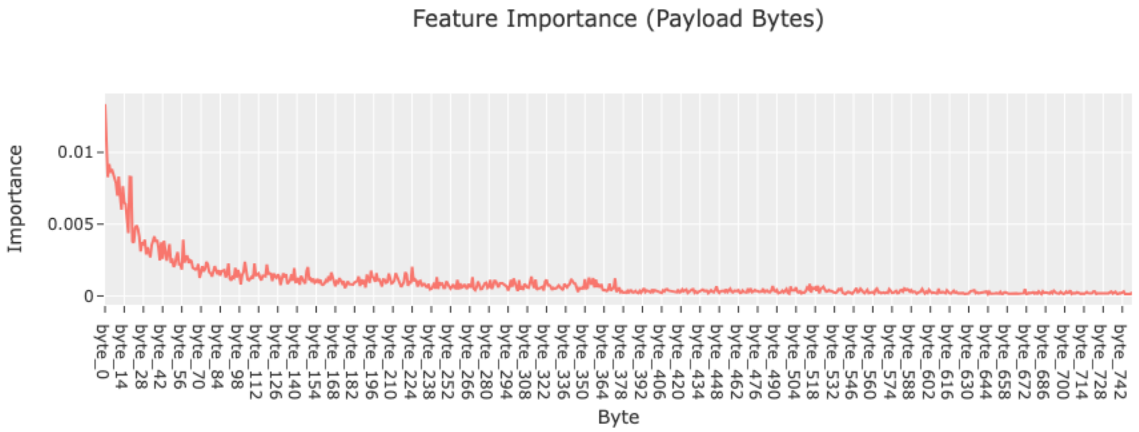


Figure 17: Importance of Payload Bytes

For the fine-tuning process, we leveraged the Hugging Face Transformers library [31] in conjunction with the PyTorch framework [34]. We loaded the tokenizer and pre-trained BERT model (distilbert-base-cased) using Transformers library, with the modeling task of sequence classification. We used the combined dataset by merging the datasets created from the UNSW-NB15 and CIC-IDS2017 datasets. We split the dataset into training and testing subsets to ensure proper evaluation and assessment of the model's performance. The training data constitutes 70% of the dataset, while the remaining 30% is allocated for testing purposes at various later stages of our research. Given that some labels in the dataset contain fewer packets than others, we encountered class imbalance, which can impact the model's performance. We applied a combination of oversampling and undersampling techniques to address this issue. Oversampling was employed for minority classes, generating additional samples to balance their representation in the training data. Conversely, undersampling was applied to the majority classes, reducing their representation to maintain a consistent distribution across all classes. This approach ensured that each class in the training dataset contained an equal number of 100,000 packets, resulting in a total of 2.4 million packets for training.

To ensure proper evaluation and monitoring of the model's performance, we further split the training dataset into a 70% training subset and a 30% evaluation subset. For efficient processing and tokenization of the packets, we employed batch processing. Each packet was tokenized using the BERT tokenizer, which generated input_ids and attention_mask for each tokenized sequence which are used to fine-tune the model. The distilbert-base-cased model contains over 65 million trainable parameters. To accelerate the training process, we leveraged the computational power of an NVIDIA A100 GPU. With this setup, we fine-tuned the model using a training batch size of 64 and an evaluation batch size of 128. The fine-tuning process involved training the model for three epochs, where each epoch represents a complete iteration through the training dataset. The training duration lasted approximately 18 hours, taking advantage of the computational efficiency of the GPU. The training loss achieved a value of 0.03385, indicating the model's effectiveness in minimizing errors and discrepancies during the training process. The evaluation accuracy reached the value of 0.9963 and the evaluation f1-score, which considers both precision and recall, also achieved a value of 0.9963. We have also evaluated the model using the test data which contains more than 500,000 packets, for which we got an accuracy of 0.9948 and micro and macro f1-score being 0.9948 and 0.9846, respectively.

Once the fine-tuning process is completed, we package the resulting model, including its configuration file, learned weights, training arguments, and tokenizer configuration files. These files are then uploaded to the Hugging Face model repository, making the fine-tuned model accessible to us and the wider community through the Hugging Face Transformers library. This saves time and computational resources, making it more accessible to leverage the model for future tasks.

3.5 Cluster Analysis

The next step is clustering the embeddings of the packets. We first initialized our fine-tuned model and tokenizer using the Hugging Face Transformers library. This allows us to leverage the capabilities of the model to generate embeddings for all the packets in our training dataset, which consists of 1.19 million packets. The process of generating embeddings follows the flow as illustrated in Figure 16. We feed the packet string to the tokenizer, which tokenizes the input and returns the tokens, input_ids, and attention_mask. These tokens represent the individual components of the packet. Next, we pass the input_ids and attention_mask to the model and retrieve the output. The output contains the last_hidden_state which represents the sequence of hidden states from the last layer of the model. The last_hidden_state contains the embeddings for all the tokens of the packet including the special tokens like "CLS" and "SEP". These embeddings are 768-dimensional vectors that capture each token's contextualized meaning and representation.

Once we obtained the embeddings for all the tokens in each packet, we further processed these 768-dimensional vectors to derive a single, representative embedding for each packet. To accomplish this, we took the mean of the token embeddings, resulting in a single 768-dimensional vector that captures the overall representation of the packet, as shown in Figure 18. To facilitate the clustering process, we transformed these final packet embeddings into 768 features, where each feature represents a dimension in the embedding space. We used standardization to transform the features to have a mean of zero and a variance of one. With these normalized features, we applied K-means clustering [38] to group the packets based on their similarities in the form of Euclidean distance. We aimed to choose a value for k (number of clusters) that is close to the square of the total number of class labels (n^2). Given that we had a total of 24 class labels (n=24), we explored various values of k, specifically those in the proximity of 576 (24^2). We employed two evaluation metrics to select the most appropriate value for k: the Silhouette score and the Davies-Bouldin score. The Silhouette score measures the cohesion and separation of the clusters, with higher values indicating better-defined clusters. The

Davies-Bouldin score assesses the cluster quality based on the within-cluster scatter and between-cluster separation, with lower values indicating better clustering results.
By examining the Silhouette score and Davies-Bouldin score for different values of k using a sample of 70,000 packets, we identified the best k value of 516 that yielded optimal clustering performance. This selection was made based on the highest Silhouette score (0.1812) and the lowest Davies-Bouldin score (1.8695), indicating well-defined clusters with minimal overlap and good separation between clusters.

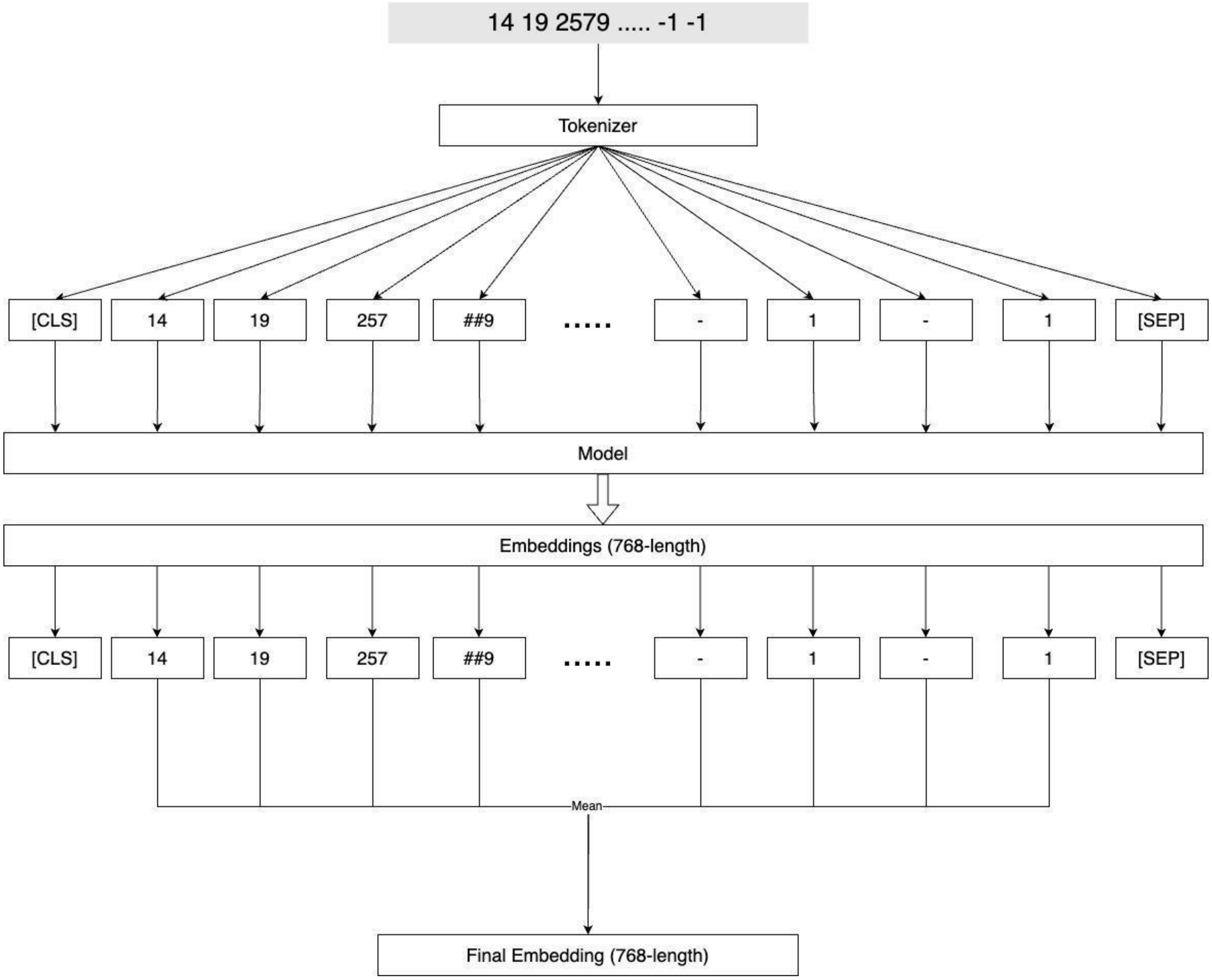


Figure 18: Embedding Generation Flow of a Network Packet

We started the clustering process with the selected value of k, for all the 1.19 million packets, which took around 20 hours to complete. It required 238 iterations for the clustering algorithm to converge and assign packets to their respective clusters. To provide meaningful names for each of the clusters, we employed a combination of manual analysis and automated techniques. We initially examined the composition of each cluster by identifying the class labels and the number of packets associated with each label contained within the cluster. Furthermore, we leveraged cosine similarity to establish relationships between the centroids of different clusters. By comparing the cosine similarity between each cluster's centroid and the centroids of other clusters, we determined the degree of similarity between clusters. We set a threshold of 0.97 to define clusters with high similarity. If the cosine similarity between two cluster centroids exceeded this threshold, we assigned similar names to those clusters. Using this approach, we systematically assigned names to all 516 clusters based on the class labels and packet compositions within each cluster. This naming process ensured that clusters with similar characteristics or packet distributions were grouped together, facilitating a better understanding of network attack patterns and behaviors.

In addition to assigning names, we wanted to generate possible network packet attacks or behaviors associated with each cluster. We employed a combination of manual analysis and the "gpt-3.5-turbo" model through the OpenAI API to achieve this [35] [36]. The "gpt-3.5-turbo" model is a text generation model optimized for chat completion capabilities and is also the primary model used by ChatGPT [37]. By feeding the cluster name and the class labels of the packets it contained to the "gpt-3.5-turbo" model with the suitable prompt, we have generated five possible network packet attacks or behaviors related to each cluster. This approach allowed us to explore potential attack scenarios and behaviors based on the composition of each cluster. To preserve the results of the clustering process, including the cluster centroids, assigned names, and the generated possible attacks or behaviors, we saved this information for further use and analysis.

3.6 Tag Generation

In order to generate tags for the packets, we required a large text corpus specifically focused on the network packet and network attacks domain. While many text datasets are available covering various domains, finding one that aligns precisely with our requirements posed a challenge. It was crucial to have a diverse range of attack information within the corpus to enable effective tag generation. Given the difficulty in finding an existing corpus that met our specific needs, we decided to create a custom text corpus using the "gpt-3.5-turbo" model through the OpenAI API. This text generation model allowed us to generate high-quality text based on prompts provided to the model. We leveraged the model to generate one paragraph for each cluster name assigned in the previous step. These paragraphs served to provide an explanation for the cluster names.

Furthermore, we generated paragraphs for each network packet attack identified in our earlier analysis. These paragraphs elaborated on the nature and characteristics of the specific attack, providing insights into its behavior and potential impact. The generation process resulted in over 250 different network attack variations, with each attack having a dedicated paragraph explaining its details. Collectively, these paragraphs formed the foundation of our custom text corpus, specifically curated for network packet attack and behavior descriptions. The final text corpus consisted of 3,010 paragraphs, each offering a unique perspective on a specific network packet attack or behavior. This comprehensive corpus encompassed a wide range of attack variations, enabling us to generate accurate and informative tags for the packets. By utilizing the custom text corpus in conjunction with the packet information and clustering results, we were able to generate meaningful tags that described the nature, characteristics, and potential threats associated with each packet.

To process the text corpus and generate contextualized embeddings for the n-gram words, we utilized our BERTSimilar Python package. This package facilitated the parsing of the text corpus and allowed us to generate embeddings using our fine-tuned BERT model.

We instantiated the class provided by the package, configuring it with the necessary parameters. These parameters included specifying our fine-tuned model as the model to be used for generating the embeddings and the same standard scaler that was used to normalize the embeddings in the clustering process. Using the load_dataset method provided by the package, we loaded the text corpus into memory and initiated the process of generating word embeddings. The package efficiently processed the corpus and produced embeddings for a total of 868,533 n-gram words extracted from the paragraphs. These n-gram words represented various combinations of consecutive words within the text corpus. Figure 19 illustrates the distribution of the n-gram words extracted by the method.

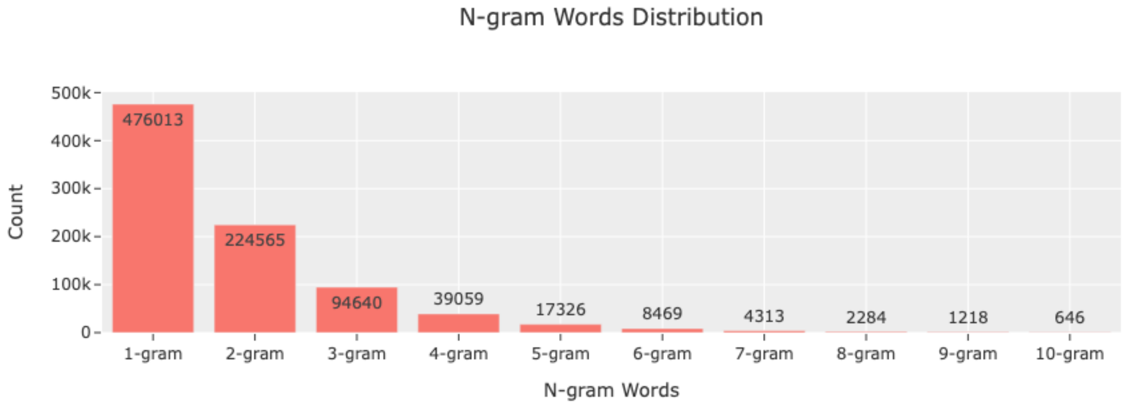


Figure 19: Distribution of N-gram Words in the Text Corpus

We utilized the "find_similar_words" method from the package to obtain similar words based on a given set of words. This method allows us to retrieve similar words to a given set of input words. In addition to returning similar words, the method also provides the embedding associated with input words that was used to find similar words. We have applied this method to the cluster names assigned in the clustering step, as well as five attacks or behaviors per cluster. By inputting these words, we obtained their respective embedding in the embedding space.

In the process of generating the embeddings, we adjusted certain parameters of the "find_similar_words" method to customize its behavior. Specifically, we set the "context_similarity_factor" to 0 and the "output_filter_factor" to 1. The "context_similarity_factor" controls the selection of paragraphs from the corpus that are considered for finding similar words. The "output_filter_factor" determined the leniency in the output words, allowing for a more flexible and inclusive selection of similar words. Leveraging the capabilities of the "find_similar_words" method, we successfully generated embeddings for all 516 clusters. These embeddings captured the nuances and semantic relationships associated with each cluster, enabling us to map them with the embeddings of the packets.

Having obtained 516 embeddings for each cluster (the centroid) and 516 embeddings for the cluster names/possible attacks/behaviors, we now have information to locate the clusters in the embedding space and associate relevant words with each cluster. The centroids, representing the embeddings of the clusters, serve as reference points that help determine the location of each cluster within the cluster embedding space. On the other hand, the embeddings generated for the cluster names/possible attacks/behaviors provide insights into the words and concepts associated with each cluster. Although the cluster embedding space and word embedding space are not identical, both sets of embeddings are created by the same model and have undergone normalization, making them comparable. We used the Euclidean distance metric to assign a given packet to the most suitable cluster. By calculating the Euclidean distance between the packet's embedding and the embeddings of the cluster centroids, we can determine which cluster is the closest match for the packet. Furthermore, by leveraging the embeddings of the cluster names/possible attacks/behaviors, we can map the packet's assigned cluster to the corresponding embeddings of words related to that cluster. This mapping enables us to get words as tags that are located closest to the embedding.

To assign tags to a given input packet, we follow a step-by-step process that involves generating embeddings, calculating distances, and selecting the most relevant tags based on the closest cluster centroid:

1. The input packet information is fed into the fine-tuned BERT model to generate its corresponding embedding. This embedding is the contextualized representation of the packet, capturing its semantic meaning and relationships.
2. Next, we calculate the Euclidean distance between the packet's embedding and the embeddings of all cluster centroids. This distance measurement quantifies the similarity between the packet and each cluster.
3. Based on the calculated Euclidean distances, we select the cluster that has the closest centroid to the packet's embedding. This cluster serves as the target cluster for tag assignment.
4. Once the target cluster is determined, we retrieve the embedding associated with the text describing the cluster. This embedding represents the contextual information and characteristics of the cluster.
5. We select the location in the text embedding space that is the same distance away from the text embedding of the cluster as the distance between the input packet embedding and its closest cluster embedding. This relationship formula is used:

   new_embedding = text_embedding + (input_embedding – cluster_embedding)
6. Finally, we utilize the modified embedding to identify the words that are closest to it in the word embedding space using the find_similar_words method. These closest words serve as the tags for the packet, describing its characteristics, behavior, or potential network attacks.

By following this process, we can generate tags for the input packet that are closely related to the cluster it is assigned to. This approach leverages the fine-tuned BERT model's contextualized embeddings, the Euclidean distance metric, and the word embeddings to ensure that the generated tags reflect the characteristics and nature of the packet.

3.7 Text Generation

Once the tags for the network packets have been generated, we can utilize them to generate descriptive sentences that provide more context and understanding. These tags are essentially words that can be used in various Natural Language Processing (NLP) tasks. To achieve this, we fine-tuned a text generation model specifically for generating text that describes the packet's header information and payload. The model is also capable of generating text that explains the tags, further enhancing the understanding of the packet. The model can provide detailed descriptions of the packet header fields, such as the source and destination IP addresses, port numbers, protocol information, and any other relevant details. Additionally, if there is payload information available, the model can generate text that provides insights into the content and purpose of the payload. Furthermore, the model can generate text that explains the tags assigned to each packet. This helps to provide a deeper understanding of the network packet's behavior and characteristics, allowing for more comprehensive analysis and interpretation.

We utilized the "Falcon-7B" model introduced in March 2023, which is an advanced language model with approximately 7 billion parameters. This model belongs to the Falcon family of open-source state-of-the-art language models, developed by the Technology Innovation Institute in Abu Dhabi. The Falcon models, including Falcon-7B and Falcon-40B, are recognized as top-performing models in the Open LLM Leaderboard for text generation tasks [20]. Falcon models incorporate multi-query attention [40], a variation of the traditional multi-head attention mechanism. In the standard multi-head attention scheme, each attention head operates independently with its own set of queries, keys, and values. However, in multi-query attention, a shared set of keys and values is utilized across all attention heads, while each head has its own query. This approach reduces redundancy and enhances computational efficiency without compromising the model's ability to capture complex relationships and dependencies within the input.

Fine-tuning models with billions of parameters presents significant technical and computational challenges. To address this, we employed the Parameter-Efficient Fine-Tuning (PEFT) methodology [41] in combination with the Quantized Low-Rank Adapters (QLoRA) approach [42]. PEFT techniques enable efficient adaptation of pre-trained language models (PLMs) to specific downstream tasks by fine-tuning only a subset of the model's parameters, rather than the entire model. This approach helps reduce computational requirements while maintaining high performance. QLoRA specifically addresses memory efficiency during fine-tuning by utilizing a 4-bit NormalFloat (NF4) data type for efficient storage and computation of weights in memory. Also, double quantization is applied, allowing the quantization constants themselves to be quantized, and paged optimizers are implemented to manage memory spikes effectively, ensuring smooth and efficient fine-tuning.

For the fine-tuning process, we utilized a dataset consisting of packet information, extracted words from the payload, and generated tags for 50,000 packets. To create the descriptive text and explanations, we leveraged the "gpt-3.5-turbo" model available through the OpenAI API, combined withmanual review for refinement. Although it is possible to fine-tune the gpt-3.5-turbo model for this specific task, this model is not open-source and is commercialized. The LoRA hyperparameters [43] used were lora_alpha = 16, which controls the initialization scale of the loralib.linear module, lora_dropout = 0.1, which represents the dropout rate in lora.linear, and lora_r = 64, indicating the rank of the lora parameters. A smaller value for lora_r reduces the number of parameters in the lora module. During fine-tuning, we employed the "paged_adamw_32bit" optimizer with various settings. The learning_rate was set to 2e-4, max_grad_norm to 0.3, warmup_ratio to 0.03, and lr_scheduler_type to "constant". These settings help optimize the training process and control the learning rate schedule. To efficiently utilize computational resources, we set the per_device_train_batch_size to 4 and used gradient_accumulation_steps of 4. This configuration allowed us to process the training data effectively. The training was performed using an NVIDIA A100 GPU, and the entire process for 80,000 samples was completed within 5 hours.

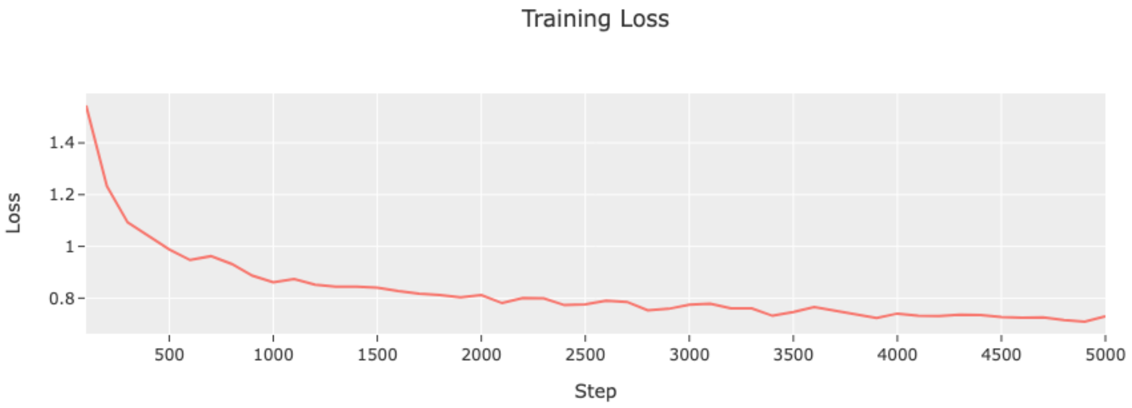


Figure 20: Training Loss over Steps of Falcon Model

The model achieved a final loss of approximately 0.7297 during the fine-tuning process. The cumulative average loss over all training steps was recorded as 0.8274, indicating the overall progress and convergence of the training process. Figure 20 provides a visual representation of the training loss per step, illustrating the gradual improvement and optimization of the model over time. To facilitate future use and accessibility, we have uploaded the model adapter configuration files to the Hugging Face Hub. This allows us to easily access and utilize the fine-tuned Falcon model for inference, equipped with the PEFT and LoRA adapters. By leveraging these advancements in model configuration and optimization, the Falcon model is now well-suited for generating descriptive text and explanations for network packet information and providing insights into the generated tags.

Chapter 4: Results

Tags provide a way of explaining network packets and also a way of using transformer models in the network security domain. To generate descriptive tags for the packets, we utilized the find_similar_words() method, which offers flexibility in generating tags with all n-gram words or specific n-gram words, as well as customization using various parameters. In order to evaluate the performance of the tag generation process, we used the test dataset that was created earlier. We sampled 50,000 packets from all class labels, and for each packet, we generated 10 tags using the find_similar_words() method.

The evaluation of tag generation performance is based on two main criteria:

1. Similarity to Original Label: We assess how similar the generated tags are to the original label of the packet. This is achieved by measuring the semantic similarity or overlap between the tags and the original label. Higher similarity scores indicate that the generated tags capture the essence of the packet's original label.
2. Relevance to Network Packets: We also evaluate the correctness of the generated tags in terms of their relevance to network packets. This involves determining if the tags accurately reflect network-related behaviors, characteristics, or potential attacks. By ensuring that the generated tags are specific to network packets, we enhance the usefulness and applicability of the tagging system.

By considering both the similarity to the original label and the relevance to network packets, we evaluated the effectiveness and accuracy of the tag generation process. This evaluation provides insights into the performance of the system and helps assess the quality of the generated tags in describing the packets' signatures and related behaviors.

We utilized "all-MiniLM-L6-v2" model [39] from the SentenceTransformers library that uses cosine similarity to measure the semantic similarity between two or more texts. We computed the similarity between each tag and the truth label or network packet linguistic to evaluate the generated tags. This analysis allowed us to assess the degree to which the generated tags were related to network security and how closely they aligned with the correct packet description. We then converted the cosine similarity scores into cosine distances, which range from 0 to 1. A value closer to 1 suggests more similarity, and closer to 0 suggests dissimilarity. The formula is given by: Cosine Distance = (1 + Cosine Similarity) / 2. Furthermore, we calculated the mean cosine distance for 10 generated tags across all packets. Additionally, we combined the tags (combination of multiple tags) and computed the mean cosine distance for this aggregated set of tags. The resulting explainability scores are presented in Figure 21. Based on the analysis, we found that the combination of all 10 tags together provided the most effective explanation for the packet. This indicates that collectively, the generated tags offer a comprehensive understanding of the packet's characteristics and behaviors, providing valuable insights into network security.

In addition to evaluating the effectiveness of the generated tags through explainability scores, we also calculated a metric called the "Variability Score". This metric measures the uniqueness and variability of the generated tags across different packets. The Variability Score is computed using the formula: Variability Score = 1 - ((Total Packets / Total Unique Labels) / Total Unique Tags). This metric provides insights into how distinct the tags are for each packet and whether there is sufficient variability in the generated tags. The results of the Variability Score analysis are presented in Figure 22. It is observed that the first tag has a lower score, indicating that it is unique to only a small percentage (15.29%) of the packets. On the other hand, the combination of all 10 tags scores the highest, suggesting that it offers the greatest variability and is the most distinct for each packet. The variability scores show a gradual increase from the first tag to the tenth tag and to the combinations of tags. This implies that the similarity, measured by cosine distance, between the first tag and the packet embedding, is usually small. In other words, the closest word in terms of cosine distance to the packet is mostly the same for all similar packets. However, as we move to the second, third, and subsequent tags, the words that are closest in terms of cosine distance start to differ more.

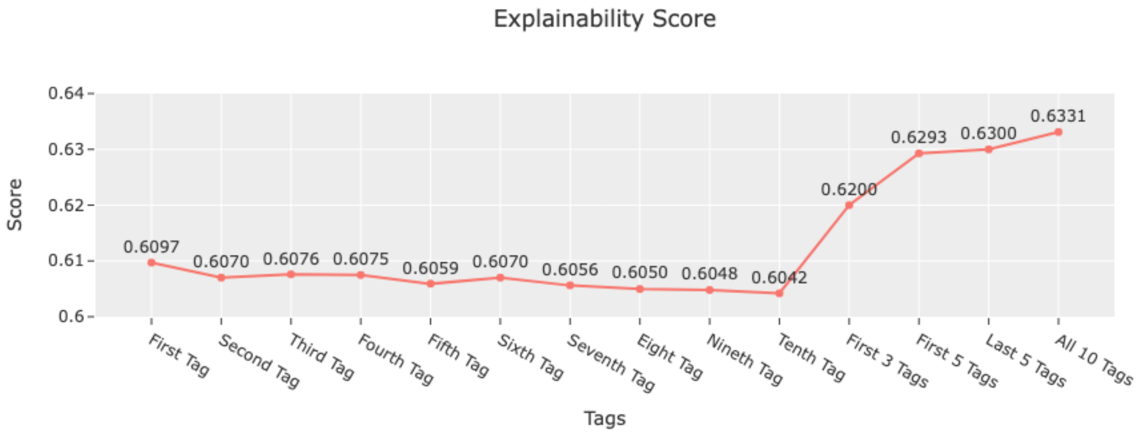


Figure 21: Explainability Score for Tag Generation

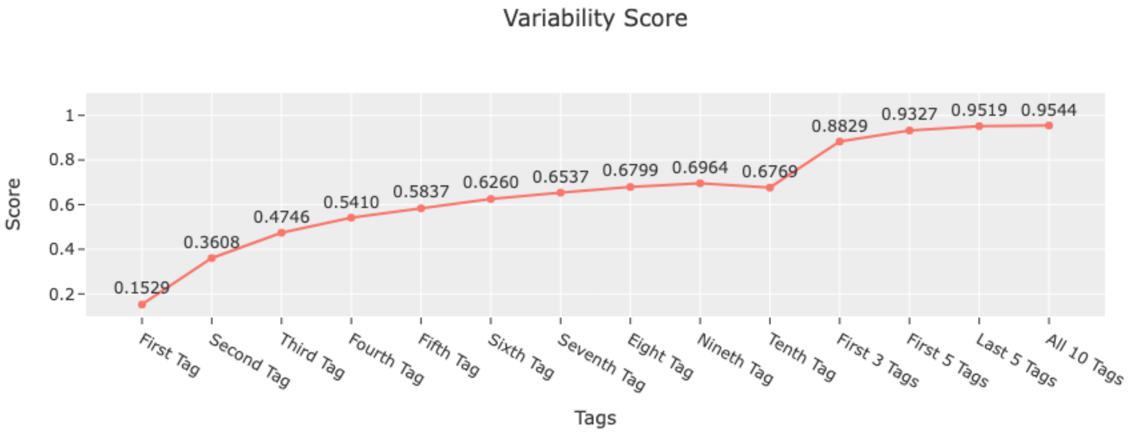


Figure 22: Variability Score for Tag Generation

By considering both the explainability scores and the Variability Score, we can gain a holistic view of the effectiveness and uniqueness of the generated tags. These metrics provide valuable insights into the quality and diversity of the tags, enabling us to assess the performance of the tag generation process and its ability to capture the nuances of the network packet data.

A comprehensive evaluation process was conducted to assess the quality and correctness of the text generated by the fine-tuned Falcon model. This evaluation aimed to compare the generated text with the correct text, focusing on the accuracy and completeness of the information provided. The "gpt-3.5-turbo" model from the OpenAI API was utilized to facilitate the evaluation, employing prompt engineering techniques to compare the generated text and the reference text. The evaluation process involved sampling 1000 packets from the test dataset, and the Falcon fine-tuned model was employed for inference. During inference, the Falcon model utilized the bfloat16 data type, which contributed to faster and more efficient inference processes. To maintain consistency and control over the generated text, certain parameters were set, such as a maximum limit of 250 new tokens and a temperature of 0 to minimize randomness in the text generation process. The correctness score was computed by assessing the fidelity and accuracy of the generated text in accurately describing the packet header fields, payload content, and the tags generated by the tag generation process. This evaluation is focused on verifying the correctness of the details provided in the generated text, ensuring that it aligns with the expected information and accurately reflects the characteristics of the packets.

The mean correctness scores for each comparison between the generated text and the reference text were computed as the next step. The overall correctness score, which represents the average accuracy and fidelity of the generated text, was determined to be 0.9235 or 92%. Specifically, when generating text to describe the packet details, the model achieved a high correctness score of 0.9553. This indicates that the generated text was able to accurately capture and convey the necessary information about the packet header fields and payload content. In terms of generating text to explain the tags, the model achieved a correctness score of 0.8872. This score indicates that the generated text was able to effectively explain the meaning and significance of the tags assigned to the packets. The high correctness scores obtained for both the packet description and tag explanation tasks validate the effectiveness and reliability of the text generation process using the fine-tuned model.

Chapter 5: Conclusion

5.1 Summary

In this research project, we have introduced a novel framework for generating tags and descriptions for network packets using transformer models. The generated tags and descriptions can be integrated with existing Network Intrusion Detection Systems (NIDS) to enhance their capabilities in classifying and analyzing network packets that are challenging to classify using traditional classification algorithms. This integration can be particularly valuable when combined with open-set recognition techniques, allowing the NIDS to handle unknown or novel network attack types effectively. The generated tags and descriptions provide valuable information about the behavior and characteristics of network packets. The network security administrators can leverage this information to gain insights into the nature of these packets and make informed decisions regarding their handling and response. These tags and descriptions can be utilized in various automatic analysis tasks and other natural language processing (NLP) applications. The structured information provided by the tags can serve as input for further analysis, such as anomaly detection, pattern recognition, and network traffic classification. Additionally, the descriptions can be leveraged for tasks such as network event summarization, report generation, and incident response.

This project involved significant data processing and analysis to extract meaningful information from the network packet datasets. We initially worked with massive PCAP files from the UNSW-NB15 and CIC-IDS2017 datasets, totaling 150 GB in size. Through a meticulous pre-processing pipeline, we transformed and cleaned the data, resulting in more than 4 TB of processed files containing packet information. The pre-processed data was organized into 108 CSV files, divided into three subsets based on different aspects of the packets. To optimize storage and improve query performance, we converted these CSV files to the Parquet file format, resulting in a final dataset size of approximately 800 GB.

The project involved fine-tuning two transformer models: BERT and Falcon. These models were trained on the curated dataset to generate tags and descriptions for the network packets. We utilized cluster analysis techniques to group packets based on their embeddings, enabling us to uncover patterns and relationships among different types of network traffic. The integration of both open-source (BERT, Falcon) and commercial (GPT 3.5) transformer models in this research showcases the usefulness and versatility of these models in the context of Network Intrusion Detection Systems (NIDS) and network packet analysis. The findings and methodologies presented in this research contribute to advancing the understanding and utilization of these models in real-world network security scenarios.

In addition to the research findings and methodologies, this project has resulted in the development of three Python packages that aim to simplify the usage and integration of the models and datasets. The "nids-datasets" package provides a convenient way to download and access the datasets created during this project. Researchers can easily retrieve the pre-processed packet and flow-level information, enabling them to seamlessly join and analyze these datasets. For generating tags, we utilized three pieces of flow-level information in combination with packet-level details, including forward and backward packets per second and bytes transferred per second. For evaluation, we introduced three custom evaluation metrics to evaluate the performance of the fine-tuned models. These metrics comprehensively assess the models' capabilities in generating accurate and meaningful tags and descriptions for network packets. The "BERTSimilar" Python package offers functionalities for generating tags using the BERT model. It provides methods to find similar words based on input words and context, facilitating the generation of informative and contextually relevant tags for network packets. Lastly, the "nids-transformers" package provides inference for the models to generate tags and descriptions that can be integrated with existing NIDS frameworks, enhancing their capabilities in identifying and classifying network attacks. The outcomes of this research provide a solid foundation for further advancements in the field, encouraging future research and development in the intersection of transformers and network security.

5.2 Limitations and Future Work

Despite the advancements achieved in this research, there are some limitations and avenues for future work that can be explored. Firstly, the fine-tuned language models, BERT and Falcon, require substantial computational resources, particularly in terms of memory and GPU capabilities. Due to performance limitations, running these models on CPUs or small GPUs may not yield practical results. For inference, using the fine-tuned Falcon model requires a minimum of 16 GB of GPU memory. The speed to generate 10 tags is 2 seconds per packet, and for text generation, it is 12 tokens per second when measured using NVIDIA A100 GPU with 80 GB of memory. Enhancements in GPU performance and efficiency would lead to faster results. The BERT model was fine-tuned to extract packet signatures and generate embeddings. Alternatively, future work could explore pre-training the model instead of fine-tuning it. However, pre-training requires substantial resources and a vast amount of data but may provide more specialized embeddings that capture the nuances of network packets and attacks. The data necessary can be easily downloaded and accessed using our "nids-datasets" Python package that contains the datasets created in this research. This research utilized TCP/IP packets and incorporated three flow-level features in combination with packet information to enhance the embeddings. Future work could expand the dataset to include other types of packets (e.g., UDP, ICMP) to create a more comprehensive representation of network traffic. Additionally, exploring additional flow-level features and their impact on embeddings and tag generation could provide further insights and improve the overall performance. Also, other language models and embedding techniques could be explored for network security applications. Investigating the performance and suitability of different models in capturing packet semantics and generating informative embeddings would contribute to a more comprehensive understanding of the network traffic. By optimizing resource requirements, enhancing model performance and speed, exploring alternative pre-training approaches, expanding the dataset and flow information, considering alternative embedding models, and ensuring accessibility of datasets, this research can serve as a foundation for further advancements in the application of transformer models in network security.

References

[1] Khraisat, A., Gondal, I., Vamplew, P., & Kamruzzaman, J. (2019). Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity*, 2(1), 1-22.

[2] Shaikh, A., & Gupta, P. (2022). Advanced signature-based intrusion detection system. In *Intelligent Communication Technologies and Virtual Mobile Networks: Proceedings of Innovations in Computational Intelligence and Computer Vision 2022* (pp. 305-321). Singapore: Springer Nature Singapore.

[3] Asad, H., Adhikari, S., & Gashi, I. (2023). A Perspective-Retrospective Analysis of Diversity in Signature-Based Open-Source Network Intrusion Detection Systems. *Research Square*. Preprint.

[4] Agrawal, V. K., & Rudra, B. (2022, December). Performance Evaluation of Signature Based and Anomaly Based Techniques for Intrusion Detection. In *International Conference on Intelligent Systems Design and Applications* (pp. 496-505). Cham: Springer Nature Switzerland.

[5] Vidal, J. M., Monge, M. A. S., & Monterrubio, S. M. M. (2020). Anomaly-based intrusion detection: Adapting to present and forthcoming communication environments. In *Handbook of Research on Machine and Deep Learning Applications for Cyber Security* (pp. 195-218). IGI Global: Pennsylvania, United States.

[6] Fu, Y., Li, H., Wu, X., & Wang, J. (2015). Detecting APT attacks: A survey from the perspective of big data analysis. *Journal on Communications*, 36(11), 1-14.

[7] Zehra, S., Faseeha, U., Syed, H. J., Samad, F., Ibrahim, A. O., Abulfaraj, A. W., & Nagmeldin, W. (2023). Machine Learning-Based Anomaly Detection in NFV: A Comprehensive Survey. *Sensors*, 23(11), 5340.

[8] Al-Bakaa, A., & Al-Musawi, B. (2021). Flow-Based Intrusion Detection Systems: A Survey. In *International Conference on Applications and Techniques in Information Security* (pp. 121-137). Singapore: Springer Singapore.

[9] Umer, M. F., Sher, M., & Bi, Y. (2017). Flow-based intrusion detection: Techniques and challenges. *Computers & Security*, 70, 238-254.

[10] Zavrak, S., & İskefiyeli, M. (2020). Anomaly-based intrusion detection from network flow features using variational autoencoder. *IEEE Access,* 8, 108346-108358.

[11] Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. *International Conference on Information Systems Security and Privacy,* 1, 108-116.

[12] Moustafa, N., & Slay, J. (2015). 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)* (pp. 1-6). IEEE.

[13] Garcia, S., Grill, M., Stiborek, J., & Zunino, A. (2014). An empirical comparison of botnet detection methods. *Computers & Security*, 45, 100-123.

[14] Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications* (pp. 1-6). IEEE.

[15] Wang, Y. C., Houng, Y. C., Chen, H. X., & Tseng, S. M. (2023). Network Anomaly Intrusion Detection Based on Deep Learning Approach. *Sensors*, 23(4), 2171.

[16] Kim, J., Kim, J., Thu, H. L. T., & Kim, H. (2016). Long short term memory recurrent neural network classifier for intrusion detection. In *2016 international conference on platform technology and service (PlatCon)* (pp. 1-5). IEEE.

[17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[18] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: System demonstrations* (pp. 38-45).

[19] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[20] Penedo, G., Malartic, Q., Hesslow, D., Cojocaru, R., Cappelli, A., Alobeidli, H., ... & Launay, J. (2023). The RefinedWeb dataset for Falcon LLM: Outperforming curated corpora with web data, and web data only. arXiv preprint arXiv:2306.01116.

[21] Rohith, R., Moharir, M., & Shobha, G. (2018). SCAPY-A powerful interactive packet manipulation program. In *2018 international conference on networking, embedded and wireless systems (ICNEWS)* (pp. 1-5). IEEE.

[22] Farrukh, Y. A., Khan, I., Wali, S., Bierbrauer, D., Pavlik, J. A., & Bastian, N. D. (2022). Payload-byte: A tool for extracting and labeling packet capture files of modern network intrusion detection datasets. In *2022 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)* (pp. 58-67). IEEE.

[23] Asgari-Chenaghlu, M. (2017). *Word Vector Representation, Word2vec, Glove, and Many More Explained* (Doctoral dissertation, University of Tabriz).

[24] Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532-1543).

[25] Xiong, Z., Shen, Q., Xiong, Y., Wang, Y., & Li, W. (2019). New Generation Model of Word Vector Representation Based on CBOW or Skip-Gram. *Computers, Materials & Continua*, 60(1).

[26] Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research,* 9(11).

[27] Řehůřek, R., & Sojka, P. (2010). Software framework for topic modelling with large corpora. In *Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks* (pp. 46-50). Valletta, Malta: University of Malta.

[28] Ferrell, B. J. (2023). Fine-tuning strategies for classifying community-engaged research studies using transformer-based models: Algorithm development and improvement study. *JMIR Formative Research*, 7, e41137.

[29] Howlader, P., Paul, P., Madavi, M., Bewoor, L., & Deshpande, V. S. (2022). Fine Tuning Transformer Based BERT Model for Generating the Automatic Book Summary. *International Journal on Recent and Innovation Trends in Computing and Communication*. 10. 347-352.

[30] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108.

[31] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2019). Huggingface's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771.

[32] Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150.

[33] Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., ... & Ahmed, A. (2020). Big bird: Transformers for longer sequences. *Advances in neural information processing systems,* 33, 17283-17297.

[34] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

[35] Ye, J., Chen, X., Xu, N., Zu, C., Shao, Z., Liu, S., ... & Huang, X. (2023). A comprehensive capability analysis of gpt-3 and gpt-3.5 series models. arXiv preprint arXiv:2303.10420.

[36] Aljanabi, M. (2023). ChatGPT: Future directions and open possibilities. *Mesopotamian journal of Cybersecurity*, 2023, 16-17.

[37] Liu, Y., Han, T., Ma, S., Zhang, J., Yang, Y., Tian, J., ... & Ge, B. (2023). Summary of chatgpt/gpt-4 research and perspective towards the future of large language models. arXiv preprint arXiv:2304.01852.

[38] Ahmed, M., Seraj, R., & Islam, S. M. S. (2020). The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 9(8), 1295.

[39] Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., & Zhou, M. (2020). Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in Neural Information Processing Systems*, 33, 5776-5788.

[40 Shazeer, N. (2019). Fast transformer decoding: One write-head is all you need. arXiv preprint arXiv:1911.02150.

[41] Mangrulkar, S. O. U. R. A. B., Gugger, S., Debut, L., Belkada, Y., & Paul, S. (2022). PEFT: State-of-the-art parameter-efficient fine-tuning methods. GitHub.

[42] Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. arXiv preprint arXiv:2305.14314.

[43] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2021). Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685.

[44] Jain, S. M. (2022). Hugging face. In *Introduction to Transformers for NLP: With the Hugging Face Library and Models to Solve Problems* (pp. 51-67). Berkeley, CA: Apress.

Appendix A: Usage of nids-datasets Python Package

```
# Install the Python Package
pip install nids-datasets
```

```
# Import the Modules
from nids_datasets import Dataset, DatasetInfo
```

```
# Show the Packet and Label Content of each 18 Files, as in Table 1
df = DatasetInfo(dataset='UNSW-NB15') # or CIC-IDS2017
df
```

```
# Details of Files to Download

# Dataset Information
dataset = 'UNSW-NB15' # or CIC-IDS2017

# Subset Information
## Network-Flows - Contains Flow Information
## Packet-Fields - Contains Header Information
## Packet-Bytes - Contains Packet in Bytes (0-255)
## Payload-Bytes - Contains Payload in Bytes (0-255)
subset = ['Network-Flows', 'Packet-Fields', 'Payload-Bytes'] # or
'all' for all subsets

# File Information
## Choose from 18 Files
files = [3, 5, 10] # or 'all' for all files
```

```
# Initialize the Dataset Module
data = Dataset(dataset=dataset, subset=subset, files=files)
```

```
# Download the Files
## UNSW-NB15 Files will be saved in UNSW-NB15 Folder
## CIC-IDS2017 Files will be saved in CIC-IDS2017 Folder
data.download()
```

```
# Directory Structure of Downloaded Files

UNSW-NB15
│
├───Network-Flows
│   └───UNSW_Flow.parquet
│
├───Packet-Fields
│   ├───Packet_Fields_File_3.parquet
│   ├───Packet_Fields_File_5.parquet
│   └───Packet_Fields_File_10.parquet
│
└───Payload-Bytes
    ├───Payload_Bytes_File_3.parquet
    ├───Payload_Bytes_File_5.parquet
    └───Payload_Bytes_File_10.parquet
```

```
# Merge the Downloaded Subsets. The merged file contains all the
information (flow, header, bytes) for each packet.
data.merge()
```

```
# Directory Structure after Merge

UNSW-NB15
|
├───Network-Flows
|   └───UNSW_Flow.parquet
|
├───Packet-Fields
|   ├───Packet_Fields_File_3.parquet
|   ├───Packet_Fields_File_5.parquet
|   └───Packet_Fields_File_10.parquet
|
├───Payload-Bytes
|   ├───Payload_Bytes_File_3.parquet
|   ├───Payload_Bytes_File_5.parquet
|   └───Payload_Bytes_File_10.parquet
|
└───Network-Flows+Packet-Fields+Payload-Bytes
    ├───Network_Flows+Packet_Fields+Payload_Bytes_File_3.parquet
    ├───Network_Flows+Packet_Fields+Payload_Bytes_File_5.parquet
    └───Network_Flows+Packet_Fields+Payload_Bytes_File_10.parquet
```

```
# The Downloaded Files can be read into the Pandas DataFrame
import pandas as pd
df = pd.read_parquet('UNSW-NB15/Network-Flows+Packet-Fields+Payload-
Bytes/Network_Flows+Packet_Fields+Payload_Bytes_File_10.parquet')
```

```
# The files in Packet-Bytes and Payload-Bytes subset contain only the
first 1500-1600 bytes
```

```
# To get all the bytes (up to 65535 bytes), use the Bytes() method

# Set packet=True for Packet-Bytes and payload=True for Payload-Bytes
data.bytes(payload=True, max_bytes=2500)
```

```
# Directory Structure after Bytes Extraction

UNSW-NB15
│
├───Network-Flows
│   └───UNSW_Flow.parquet
│
├───Packet-Fields
│   ├───Packet_Fields_File_3.parquet
│   ├───Packet_Fields_File_5.parquet
│   └───Packet_Fields_File_10.parquet
│
├───Payload-Bytes
│   ├───Payload_Bytes_File_3.parquet
│   ├───Payload_Bytes_File_5.parquet
│   └───Payload_Bytes_File_10.parquet
│
├───Network-Flows+Packet-Fields+Payload-Bytes
│   ├───Network_Flows+Packet_Fields+Payload_Bytes_File_3.parquet
│   ├───Network_Flows+Packet_Fields+Payload_Bytes_File_5.parquet
│   └───Network_Flows+Packet_Fields+Payload_Bytes_File_10.parquet
│
└───Payload-Bytes-2500
    ├───Payload_Bytes_File_3.parquet
    ├───Payload_Bytes_File_5.parquet
    └───Payload_Bytes_File_10.parquet
```

Appendix B: Sample Input and Output of Tag and Text Generation

```
# Install the Python Package
pip install nids-transformers
```

```
# Import the PADEC Module
from nids_transformers import PADEC
```

```
# Initialize the Models and Tokenizers
padec = PADEC()
```

```
# Flow Information (From Wireshark Conversations Window)
forward_packets_per_second = 0
backward_packets_per_second = 4
bytes_transferred_per_second = 5493
```

```
# Packet Data in Hexadecimal (From Wireshark Hexadecimal View. Copy as
Hex Stream)
packet_hex = '''
3ca6f60849b920b39957e74b0800450005c881dc0000f506c2790d235d2b86588b3301
bbf95a94eccbfa554bbac980100085d54400000101080abcb794b10c6ab7722057d826
13cc2c721b879ef00e6d925bca92a02d529fd587fd8e5a9cb93dd2a405d8315612500d
7179cf7c01ca5e18cd137fe2044fe15898d5b42722f9e79bbc7431ce711171aa63a6b7
79367d745a0b5432fa326e8e7238d15033da601a4bb9c9bea464f6ca54b64698f31493
d9da42fa6e0904a15fb1f944b96de8c55909f7e8780be2de10786b0ff623e503f94276
a694bbf823686654ebcdafbfce9f5677e3d21ac1d25426a2be1badeadc5f29449a0244
19bba4d350ce7494563e9dabaa2c405e21a5fc918586193499139bd967d06ad188e844
6ce0ddd406a336847bb64e1e70a73aaffdd1fdfc8cddd89b73433fe0fdcfe11dffa208
710e0ecec840b632071872bb688353f59740f45d1efec153e2cc2b69f756b871073a8a
f9ca923eb213df7c1a67f5679d64e3e758394695fa486c32fd43d454bacc5b5f733eb5
e28f70d605ff0947cf68e27dd51081b08ee083976d6b6eb277bd5e8787cb80e0bd574b
6f6493e626999467e098ec329fd049d7d20ddc18547e2284e5560509692ce6e86fee5e
ce2997757697279dbbe418c37a86a79829b34cf8cb52e07e389c61373eff20705d8906
aa6d98d5169bb316e963c6a85c8a4f5aea12d6e9a5402cb2aacc63be2b5a845bb5be1f
416e19764f44b57837a854d233b764cbb8849f49a5c3deb77a0208cb512d973034c36d
```

```
90870efdbad00c55fc3d85ef76fd275c21cf0cfbd6cf3cebbd0c62d3c4e8cb21a65b09
83c1ed24d9f0a2bd1831316d62aeb6ec9e14a998803671b12d4dcf37151b75b69ec28c
ca72a36f67b5d3ec3f02606f94ebf941c0f705fd3ba39a154dcb20b1929df10c2ced9d
b7de3f2bfca59528e699591436b605ae5c174e3c3d7a237c72a0cce22d4cc370767d78
a7ed485eb5fc96f6ae45e7e3114ecb1aab59acdcc14a7303b4f49484c2b834f8289e00
6bd4c6ae38018db9c48ea09caa095b25a0e626486713e07ca409ff52918d6bd390903d
b3b3a5f823cb91dab2d515c34f459c58dd242529322bc10428786451bd7c2d899f0398
c9ffc37302b0d2dca95569d29db478705ed7c85a27ec00cb827c4671424ee33a49a80e
c1e63b3a810af84ea42bdac72b6c9a5aa5438bdc4461a9bf3dafc67645707291 8c6c6a
65aaed79a1be272f006edf7c2e930919a53a2eae0749d98cdd9c1b482d4db4adb7a986
5ac613bb9a9d8110a72f3f4f40a58fe9fa8eec36e1eee61124d84e92001c617fb025e4
8e250a173e031552575b48e67d67c988c432364e945e5b3845d61090ccbb628504aac0
d453a91c75fa23d6d59b65eadfe79c10f9878715780b9c5b68df37234ddd723b002361
1c647f17fddaf0266eec2faa7e745fb06017cbcba1608fd3a9903036d3c5505a3185d0
b31f512106509a4cc5582fe13283a18d817b95feb25a61782f2a571722c24979fb39ef
af823be465483271e4c4dcc39a8cbc930492ed1b224aa37c50dc19e67b4f1117f92d0b
d6ef81cbc72ac2189e27d893b838a19d7a2b8a9b46a6786fdbcfa3749cf564b0038440
418a7c9fe2f477458ef743270aeafe0bf510f043a7e7d54787ab92ba80f97d75e06f4b
c25cb521d54d221fd089d408d7c9166268376c5c2de1c2f44dc6c0402c35a0f55b2f3e
a13f80a11a80f65d41bcb63dac7ae9cfa063a8c749231d6d2cd9b5a83252972f0dd424
efa79b72bf558d1648dd2c78c202e7398eef6b8adeab334227e92534e7f3dd26bdaa85
6ce1feba77f87005e4ed87a6dae4c2bb2c72eecfaaf9e1299cb2f0ff1f3f8cff459e30
396bf595d7c08a9a704a394211cc459e01a939cb6cbf8627ceefebb1b338d47079e395
8009d2388b86e38a9a5c51f2134c304f98c21d00951c8aa15d3f47e9ba61fa43606d91
698000bb7427365ef8b485d11bcdfcea0d52e40af2b76e9f3d372b15c9463b18660f23
cd5f04e660f727467a34d8994b22f713f1bfaaf2cb1a0b2aaaa3b1caacd6955ec3e96f
de2ca82b5caedc45521cb3978a7c3d65b4076ec96f069608'''
```

```
# Generate 10 Tags
tags = padec.GenerateTags(packet_hex_stream=packet_hex,
       forward_packets_per_second=forward_packets_per_second,
       backward_packets_per_second=backward_packets_per_second,
       bytes_transferred_per_second=bytes_transferred_per_second,
       total_tags=10)
```

```
# Output
tags
```

```
{'considered regular expected': 0.9683609588041633,
 'malicious intent': 0.9615794189652873,
 'typical': 0.961300843268933,
 'reference point': 0.9590238971149821,
 'label signifies normal network behavior': 0.9575988655384243,
 'standard network protocols': 0.9566183076210247,
 'baseline': 0.9542804079487445,
 'Average': 0.9535212932036061,
 'abnormal traffic patterns enabling': 0.9528368377318245,
 'expected traffic patterns': 0.952349645227717}
```

```
# Generate Text explaining the Tags with a maximum of 250 Tokens
text = padec.GenerateText(explain_tags=True, max_new_tokens=250)
```

```
# Output
text
```

```
Based on the majority of the tags, the network packet appears to be a
normal packet. It is considered regular and expected, with typical
behavior and standard network protocols. The label signifies normal
network behavior, and the packet follows expected traffic patterns.
There is no indication of malicious intent or abnormal traffic
patterns. Overall, it is a regular and normal packet. However, further
analysis may be required to ensure its security.
```

```
# Generate Text explaining the Packet with a maximum of 250 Tokens
text = padec.GenerateText(explain_packet=True, max_new_tokens=250)
```

```
# Output
text
```

```
This network packet is an IPv4 packet with a length of 1480 bytes. The
packet has a Time-to-Live (TTL) value of 245, indicating that it has
been forwarded through 245 routers. The packet is using the TCP
protocol and has a source IP address of 13.35.93.43 and a destination
IP address of 134.88.139.51. The source port is 443 and the
destination port is 63834. The packet has the ACK flag set, indicating
that it is acknowledging a previous packet. The payload of the packet
contains various words and phrases, such as "VPqy," "qqcy," "KdTFBn,"
and "DmY." These words do not provide much context or meaning, but
they could be part of a message or data being transmitted. Overall,
there are no abnormalities in the packet, and it appears to be a
normal TCP packet with a specific payload. However, further analysis
would be required to determine if any security issues or anomalies are
present. The packet does not exhibit any suspicious or malicious
behavior. The IP version is 4, indicating that it is an IPv4 packet.
```

Appendix C: Code (GitHub Repository Links)

BERTSimilar - Get Similar Words and Embeddings using BERT Models
https://github.com/rdpahalavan/BERTSimilar

NIDS Datasets - Download UNSW-NB15 and CIC-IDS2017 Datasets for Network Intrusion Detection (NIDS)
https://github.com/rdpahalavan/nids-datasets

NIDS Transformers - Tag Generation and Text Generation Inference for Network Packets using Transformers
https://github.com/rdpahalavan/nids-transformers

PADEC - Packet Describer (Tag and Text Generation) Source code of this Research Project
https://github.com/rdpahalavan/PADEC