POLITECNICO DI TORINO

Master's Degree in ICT for Smart Societies

Master's Degree Thesis

Designing and engineering a Q&A LLM for network packet representation

Supervisors

Prof. Luca VASSIO

Prof. Marco MELLIA

Phd. Matteo BOFFA

Candidate

Giovanni DETTORI

October 2024

Abstract

As internet traffic continues its ever-growing rapid expansion, the development of solutions for automatic analysis becomes increasingly important for ensuring network performance, security, and reliability.
Tools like Wireshark and traffic analyzers enable packet analysis in a human-friendly way by generating verbose and detailed textual descriptions. However, such detailed analysis does not scale efficiently with large datasets and faces significant challenges due to the prevalence of encrypted traffic. This necessitates the development of more sophisticated tools, helping or even substituting traditional methods, to extract meaningful information directly from the raw packet, with minimal human engineering.
Nowadays, advanced machine learning algorithms and deep learning models offer the ability to learn complex patterns and relationships within the data. This leads to the following question: can the text analysis capabilities of large language models (LLM) replace traditional tools allowing larger-scale and automatic internet packet analysis?

The thesis introduces an artificial intelligence pipeline to obtain a model that processes textual packet data, i.e., the so-called PCAP raw format, and generates self-supervised numerical representations that capture the full meaning of the original packets. This is obtained through two fine-tuning methods applied to the pre-trained T5 model. The first is a simple autoencoder while the latter emulates what the tool Wireshark already performs by asking an LLM different questions on the internet packet header and payload. A bottleneck is introduced between the encoder and the decoder of the T5 model, to obtain the numerical representations of the packet in input.

With the model evaluation, we demonstrate that our approach is feasible, enabling us to tackle networking tasks such as application classification or service recognition on par with state-of-the-art solutions. Moreover, we show that these representations can be utilized for unsupervised and scalable analysis of large datasets.

Acknowledgements

"Do not settle for what, but get to know the why and the how."
Robert Baden-Powel

I would like to express my deepest gratitude to all the remarkable individuals who have surrounded me throughout my Master's Degree journey. Their unwavering support, encouragement, and wisdom have been invaluable in the completion of it. Surrounding oneself with such dedicated and inspiring people truly makes all the difference.

Special thanks to my supervisors, Prof. Luca Vassio and Prof. Marco Mellia. Their unwavering presence and weekly support have been essential in helping me make the right choices throughout this research. Words cannot express my gratitude to my tutor and friend, Matteo Boffa, who has always supported me as soon as I was in trouble regardless of the hour, always spurring me to go in-depth into things.

I would like to express my gratitude to my large family. Their unwavering belief in me and faith in my abilities has been a constant source of encouragement through challenges and staying focused on my goals, even during the most difficult periods.

I extend my sincere thanks to all the friends I have met during my academic and private life. Each of you has played a significant role in shaping who I am today. Your support, encouragement, and companionship have been invaluable to me. Whether through shared experiences, heartfelt conversations, or simply being there during challenging times.

Finally, I am particularly grateful to my primary school mathematics teacher Anna, who spurred me to pursue a career as an engineer since I was a child.

Table of Contents

List of Tables

List of Figures

Acronyms

AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations from Transformers
BPE	Byte Pair Encoding
DAE	Denoising AutoEncoder
DL	Deep Learning
ET-BERT	Encrypted Traffic BERT
ICT	Information and Communications Technology
IDS	Intrusion Detection System
IoT	Internet of Things
LAN	Local-Area-Network

LLM	Large Language Model
ML	Machine Learning
MLM	Masked Language Modeling
MLP	Multi Layer Perceptron
NLP	Natural Language Processing
PCAP	Packet CAPture
QoS	Quality of Service
RNN	Recursive Neural Networks
SOTA	State-Of-The-Art
SVM	Support Vector Machine
T5	Text-To-Text Transfer Transformer
UMAP	Uniform Manifold Approximation and Projection

Chapter 1

Introduction

In this era of digital innovation, more and more challenges lie ahead for the world of research and business to analyze and filter internet data quickly. Indeed, in a single ICT system, computers, IoT devices, storage devices, routers, and many other components must communicate to maintain good QoS, security, and latency performance.

1.1 Problem definition

As internet traffic continues its ever-growing rapid expansion [1], the development of solutions for automatic network analysis becomes increasingly important for ensuring network performance, security, and reliability. Traditional traffic analysis methods often rely on static rules and security experts. However they are becoming less effective for the increasing complexity of network environments, the dynamicity of protocols, and the growth of encrypted traffic.
Tools like Wireshark, a well-known internet traffic analyzer, enable packet analysis in a human-friendly way by generating verbose and detailed textual descriptions. However, such detailed analysis does not scale efficiently with large datasets. This necessitates the development of more sophisticated tools, helping or even substituting traditional methods, to extract meaningful features directly from the raw packet, with minimal human engineering. Then, consistent packet representation finds application in a large number of fields, from classification and novelty detection to QoS monitoring and intrusion detection.
Nowadays, to address this problem advanced machine learning algorithms and deep learning models are leveraged for their ability to learn complex patterns and relationships within the data [2]. Furthermore, these solutions can be constantly improved and adapted to the evolving nature of internet traffic.
In the literature, several attempts exist to create a unified structure for representing

internet packets. This often involves using fingerprinting techniques, which tend to lack scalability. In contrast, more scalable solutions leverage machine learning to automate the feature extraction process but rely on fixed structures such as nPrint [3], or need end-to-end fine-tuning such as in PacRep (Packet Representation)[4] that focuses on optimizing performance for particular applications. This can limit their generalization ability and flexibility. As a result, they often fail to provide a comprehensive and scalable solution for internet packet representation. Consequently, any minor alteration in the task or amount of data makes the model obsolete, necessitating a complete retraining process.

1.2 Thesis purposes

The thesis aims to explore a possible application of the LLM in networking trying to identify a strategy to find a consistent packet representation in the internet flows that allows a scalable internet traffic analysis. The focus is to propose a training pipeline that is able to obtain a packet representation, evaluated on a set of classification tasks. An important analysis is performed on which is the best way to have a single representation starting from N vectors that come from the model's encoder. The solution is to implement a special layer called *bottleneck* that executes the compression by performing simple operations, such as mean or max, up to more complex ones, such as the Luong attention.
A secondary goal is also to provide an exhaustive analysis of the datasets used in order to give an overview of the most commonly used datasets in the literature, and the reasoning behind their use in the thesis.

1.3 Thesis contributions

The thesis work is an initial exploration of a field not widely considered, for this reason, I had to investigate and create from scratch a lot of essential elements starting from the datasets up to the final training pipeline. Among the entire contributions of the thesis, I reported the main ones in the following:

1. The proposal of a model training pipeline to obtain a numerical representation of an internet packet. It starts with two pre-training steps applied to the pre-trained T5-model. The first is an autoencoder that takes as input a packet, retrieves the internal representation through the bottleneck, and finally attempts to recreate the input packet. The second pre-train is a supervised question-answering in which the model learns to solve retrieval and reasoning tasks.

2. A possible benchmark to evaluate the quality of the packet representation. In particular, I tested the model on one task derived from the widely used dataset ISCXVPN2016 [5].

3. An in-depth description of the most spread datasets in the literature, focusing on the ones largely exploited in the thesis.

Chapter 2

Literature review and background

This chapter conducts an extensive review of the literature relevant to the thesis. The aim is not to exhaustively cover all the topics exploited in the study but to ensure an understanding of the methodology and the results obtained. After analyzing the state of the art about the thesis topic, the following sections try to make a general overview of the world of the large language models going deep into their base structures and presenting T5. Moving on, we present the basis of networking to better understand the field in which the thesis is applied.

2.1 State-of-art: packet classification

The first stages of traffic analysis saw the growth of rule-based methods based on fixed rules designed by security experts. They consisted of recognizing patterns or statistical behaviors of communication protocols and port numbers to identify possible violations of security policies [6, 7, 8]. However, rule-based techniques are becoming less and less effective for the increasing complexity of network environments, the dynamicity of protocols that can cause high false-alarm rates because the packets are not recognized, and the growth of encrypted traffic [9].

High progress in traffic analysis was achieved with the introduction of machine-learning techniques (ML), used to analyze the high-dimensional statistical features of traffic. For instance, with their work, Zamani and Movahedi [10] demonstrated that employing a machine-learning approach for intrusion detection achieves a high detection rate and a low false-positive rate while quickly adapting to evolving intrusive behaviors. In the beginning, the Naive Bayes linear probabilistic classifiers were particularly successful, exploiting many features computed on internet flows and packets [11, 12]. Other techniques were based on Support Vector Machines (SVM) and random forests. An example of the first can be the work proposed by Panchenko [13] which selects 104 optimal statistical features and feeds them to an SVM to identify website traffic. Concerning the second, Zhang [14] proposed a systematic framework that applies the Random Forest algorithm to both misuse and anomaly detection in IDS. Even if ML methods combined with statistical features, can analyze complex traffic patterns they rely heavily on statistical features crafted by experts who must identify the most suitable features for each scenario. This makes their application difficult in real-world network settings [15].

With the spread of deep learning (DL) methods, they become the main tool for automatically extracting traffic representations and achieving a significant performance improvement [2]. In 2018 Miller [16] achieved good results by applying MLP to categorize encrypted VPN and non-VPN network traffic. Wang [17] proposed CNN structures to solve malware traffic classification in which the packets are transformed into two-dimensional images.

Slightly after the increase in the use of DL, also large language models (LLM) started to be massively used in traffic classification and analysis, thanks to the introduction of the concepts of "attention" and "transformer" [18]. In this context, the work of Lin et al. [19] and the one of Zhao et al. [20] are particularly interesting. In the first work, the authors proposed ET-BERT. This model leverages the power of a pre-trained transformer to create contextualized datagram representations in order to identify and classify encrypted network traffic accurately. The second one retraces Wang's idea [17] by creating multi-level flow representation matrices and training YaTC (Yet Another Traffic Classifier), a masked autoencoder for internet traffic classification. Even if the excellent results on different challenging classification datasets these solutions do not provide an accurate packet representation. Thus, in the thesis, we try to exploit the power of the T5 model [21] to obtain an internet packet representation and only after, use it to achieve sota performance on different well-known classification datasets.

2.2 Large language models overview

Large language models (LLMs) are advanced computational models designed to understand and generate human-like text. They are trained on a huge amount of data, enabling them to perform a wide range of processing tasks such as text generation, translation, summarization, question answering, information retrieval, and conversational interactions.
LLMs use deep learning techniques, particularly transformer architecture, to learn

patterns and relationships in language. This allows them to produce coherent and contextually relevant responses with human-level performance.
The term *large* is not only due to the data needed to train such models but also to the number of parameters that make up them (a few million up to billions and trillions).

2.2.1 Working pipeline of a LLM

Since the beginning of the spread of LLM, a huge number of different architectures and types of models have been developed. Notable examples include BERT [22] from Google, GPT-3 [23] and GPT-4 [24] from OpenAI, and LLaMA [25] from Meta. Despite their differences, these models share many common elements in their underlying mechanisms. Let's explore the key components:

- **Tokenizer**: the process performed by the tokenizers is essential in the NLP pipeline since they convert text into data that models can process. Indeed, models only understand numbers, so tokenizers convert text inputs into numerical data. The process starts by splitting the input string into small manageable units, called tokens, following different rules. The rule's complexity can change depending on the model we choose. In the following the most common algorithms are reported:
  - **Byte-Pair-Encoding**: a data compression algorithm that starts by selecting individual uni-grams (letters of the alphabet) and then merges them into n-grams by selecting the most frequent. The size of the tokens increases until the maximum number of possible tokens is reached.
  - **WordPiece**: like BPE, wordpiece uses a bottom-up approach that starts with the single alphabet characters and then tries to insert groups of them choosing not only the most frequent but also the one with the highest likelihood.
  - **SentencePiece**: a widely used method that works over BPE that tries to solve the *multiple sub-word segmentation* problem. Indeed, the objective is to select tokens that appear frequently (to measure the importance of the word), but differently (to maximize the information captured).
- **Vocabulary**: the set of the independent tokens we selected best during the tokenization process. Since the *intelligent* part of the model can work only on numbers, the vocabulary contains a mapping for each token with a dense vector in a continuous vector space. The idea behind the continuous vector is to capture the semantics of the token in relationship to the other tokens.
- **Model**: the *intelligent* part of the LLM, designed to understand and generate human language. It consists of a machine learning system, trained with many

data to solve different tasks. Nowadays, its main component is the transformer which is widely explained in subsection 3.2.3.

- **Output**: the text or data generated by the model in response to a given input or prompt. The output makes it possible to evaluate the model's performance.

2.2.2 Fundamental component: the transformer

The Transformer is a new architecture proposed by Google in 2017 to replace the previous solutions used in most natural language problem applications. Its main goal is to solve tasks where the input and output are both sequences (sequence-to-sequence tasks), especially to handle the long-range dependencies among words.
The general structure of the transformer is composed of two big functional parts: the encoder and the decoder. Depending on the different implementations it is possible to increase or decrease the complexity of the two elements by developing new architectures on the original transformer modules.
Let's explore the fundamental components of transformers to gain a deeper understanding of their architecture:

- **Embedding Layer**: a type of hidden layer that maps high-dimensional input data into a lower-dimensional space. This transformation helps the network learn relationships between inputs more effectively, and process the data more efficiently. Indeed, they convert words or phrases into continuous vectors that capture semantic meaning.

- **Encoder**: It analyzes the input text to identify and extract key information, and convert it into a continuous representation which is then forwarded to the decoder. Here's a breakdown of its key functions:

  - **Self-attention mechanism**: The LLMs utilize this mechanism to discern the intricate dependencies and relationships between input sequences and the current element. It allows the model to focus on relevant parts of the input, similar to how humans pay attention to specific details when comprehending language.
  - **Feed Forward**: This component is a multi-layer perceptron, composed of at least three layers: the input, hidden, and output layers. This neural network aims to introduce a non-linear transformation inside the mode, allowing it to learn more complex patterns and relationships in the data.
  - **Layer Normalization and Residual Connections**: each layer includes normalization and residual connections to stabilize and improve the training process.

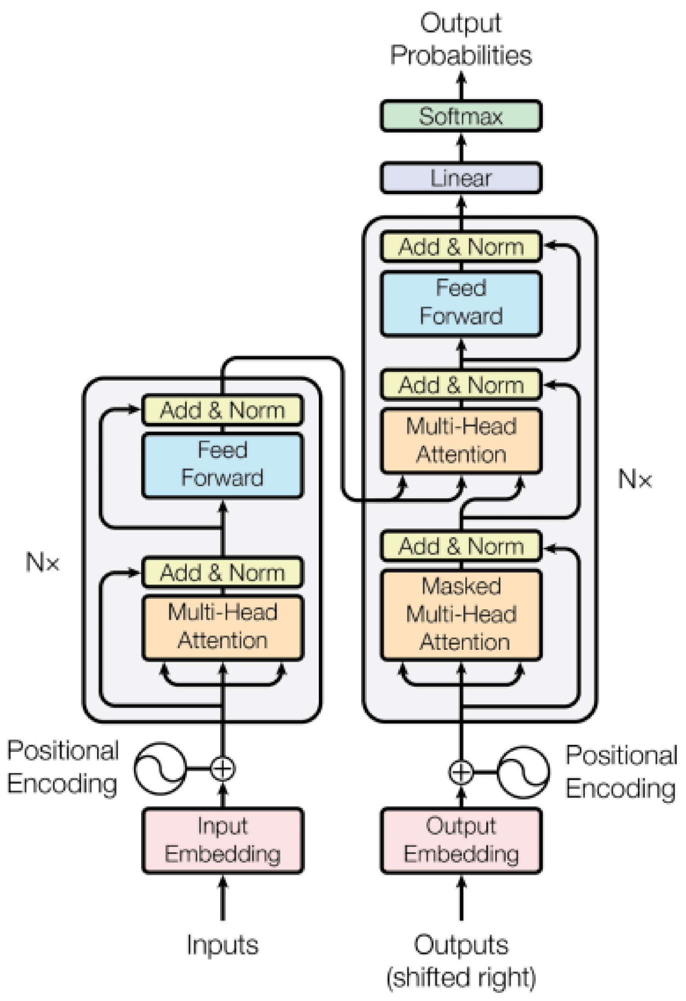

Figure 2.1: Transformer base architecture - *source* [18]

- **Decoder**: It is the part that generates the output of the model using the information extracted by the encoder. At each step, the model outputs the most suitable token. The contextualized representation from the encoder and the previous outputs are considered to generate the next token in the sequence. As the encoder, the decoder utilizes several layers, each composed of different elements some of which are shared with the encoder. Two are the new elements introduced:

  – **MultiHead-attention mechanism**: instead of implementing a single attention layer to allow the model to focus on different parts of the input sequence simultaneously, more attention layers running in parallel are used. The final output of each attention layer is then concatenated and the concatenation is linearly transformed to produce the output. This architectural choice has the dual benefits of a richer representation without increasing the computational cost. Indeed, by simultaneously focusing on various segments of the input, the model can identify more intricate relationships within the data but the overall computational cost is comparable to that of single-head attention working on the same dimensionality.

  – **Masked MultiHead-attention mechanism**: considering that the attention of the decoder must be unidirectional and the model cannot *look ahead*, this layer introduces a mask. This way, the model predicts the next word exclusively from the preceding context, as in a language flow.

- **Linear and softmax layer**: transforms the decoder's output to assign a probability to each token in the vocabulary. Afterwards, the probability can be used to choose the best token to output.

2.2.3 Self-attention

Self-attention is a mechanism especially used in Transformers because it allows the model to weigh the importance of different parts of the input data relative to each other. After the tokenization of a sentence a fixed number of tokens is passed to the model. Afterward, the query vector, key vector, and value vector are obtained from each input vector. These are calculated by multiplying the input with different weight matrices W_q, W_k, and W_v which are learned during the training phase.
The idea behind the three vectors is that the query has the property to be the "information seeker". It's like the model formulating questions to determine which words part of the sentence are most relevant to the given word. The key provides context to the words by trying to identify a contextual tag or meta-information. Finally, the value represents the inherent meaning of a word.

To better understand how self-attention works, the following is divided into phases:

1. Evaluate the "score" for each input token by performing the dot product between its query and the key vector of all the other inputs. In this way, if the input vector has ten words, every word will have ten score values.

$$score_{1,1} = q_1 \cdot k_1,\ score_{1,2} = q_1 \cdot k_2,\ ...,\ score_{1,n} = q_1 \cdot k_n$$

2. Divide each score by $\sqrt{d_k}$ where d_k is the dimension of the key vector. This operation is fundamental in stabilizing the gradient. Indeed, without the scaling factor, the dot products of the query and key vectors can become very large, especially when d_k is high.

3. Apply the SoftMax function to the scores to normalize them, ensuring all values are positive and their sum equal to 1.

4. Multiply the value vector by the scores obtained in step 3. In this way, only the most relevant word(s) will assume a high value in the output vector.

5. Sum up the vectors obtained in step 4. The computed vector is the output of the self-attention that can be used in the following layers of the transformer.

However, in the actual implementation, the calculation is done in matrix form to speed up the processing. So, the following formula summarises steps 2 to 5:

$$SoftMax\,(\frac{Q\,K^T}{\sqrt{d_k}})\,V$$

where Q, K, and V are respectively the query, key, and value matrices.

2.3 Denoiser autoencoder

A denoising autoencoder (DAE) is a model designed to reconstruct the original input from its corrupted or noisy version, using an internal low-dimensional representation of the input data. In practice, the idea is to add noise to the input data, and then during the training, the model learns how to reconstruct the noise-free version of the input.
Fundamental is the internal representation that is less sensitive to the noise. Indeed, there is the hypothesis of "*robustness to partial destruction of the input*" [26], meaning that different noise versions of the input should produce nearly identical representations. This is based on the reasoning that a good representation should capture stable structures, dependencies, and regularities characteristic of the underlying distribution of the observed input.

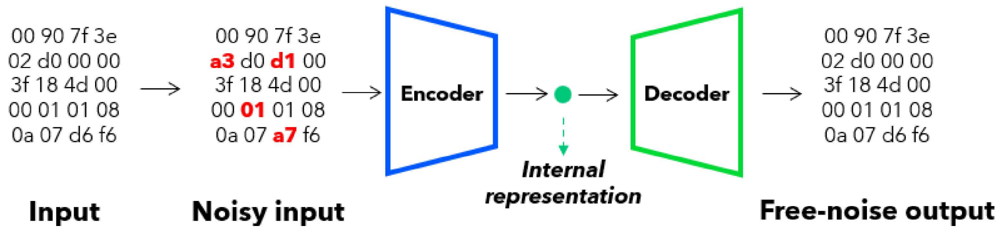

Figure 2.2: Base structure of a classification head.

2.3.1 Classification head

The classification head is the final component designed to interpret and categorize the output generated by the model. Essentially, it is a layer or a set of layers added on top of the base model to transform the high-dimensional representations produced by the model into a format suitable for specific classification tasks. As anticipated in subsection 3.2.3, the head typically consists of a fully connected layer followed by an activation function, such as softmax, which converts the raw scores into probabilities for each class.

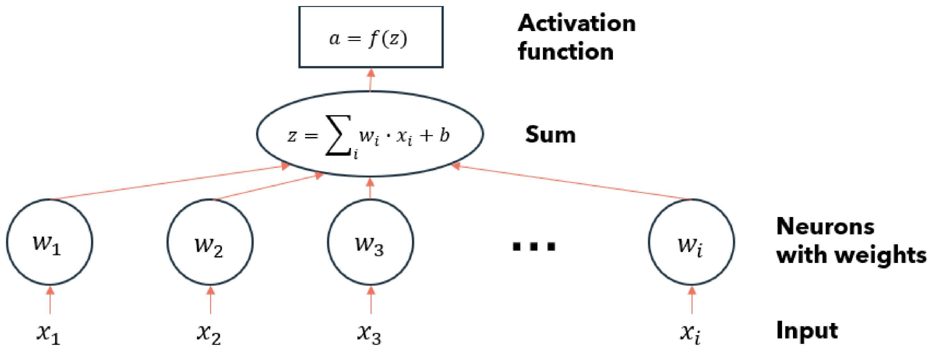

Figure 2.3: Base structure of a classification head.

Let's go deep into the two elements:

- **Linear layer**: a set of neurons (corresponding to a node in the neural network graphs) all connected with the previous and subsequent layers. As Figure 2.3 shows, each neuron receives an input (x_i) that is multiplied by the weight (w_i) associated with the neuron. If there are several inputs, a summation is performed, then a bias (b_i) is added.
- **Activation function**: a non-linear function that takes the weighted sum of the output of all neurons and generates the final output. The most common non-linear activation functions are ReLU, Sigmoid, or Tanh. The introduction of non-linearity, allows the network to learn complex patterns within the data.

2.4 T5 model

T5 (Text-to-Text Transfer Transformer) [21], launched by Google AI in 2019, is a collection of large language models trained on an extensive dataset of text.
The core concept of T5 is to address every text processing challenge as a "text-to-text" problem, meaning that, taking a text as input, the goal is to generate new text as output. The great advantage of the text-to-text framework is that enables

consistently applying the same model, objective, training procedure, and decoding process across all tasks tackled. The derived flexibility allowed the authors to assess performance across a wide range of English-based NLP tasks, such as question answering, document summarization, and sentiment classification. This unified approach allows them to compare the effectiveness of various transfer learning objectives, unlabeled datasets, and other factors while pushing the boundaries of transfer learning for NLP by scaling models and datasets beyond previous limits.

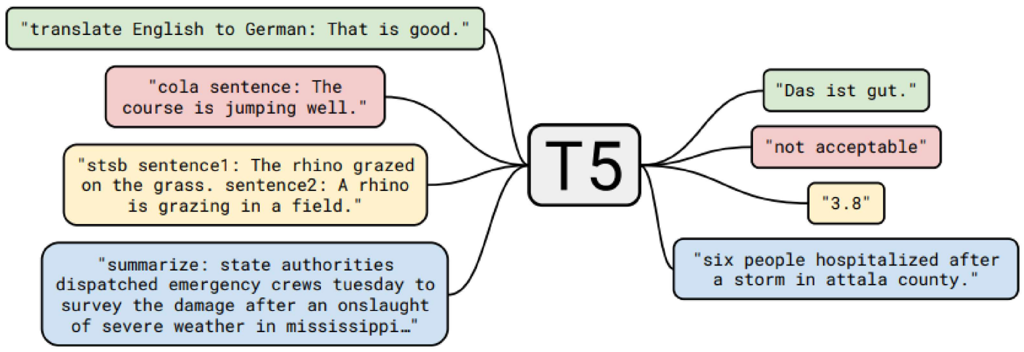
"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Figure 2.4: Diagram of T5 text-to-text framework - *source* [21].

The different models proposed in the T5 original paper, shown in Table 2.1, are based on encoder-decoder Transformers architecture. These models are distinguished by their parameters, which reflect the model's complexity and potential capacity.

Model	**N° of parameters**	n_{layers}	d_{layer}	n_{heads}
Small	60M	6	512	8
Base	220M	12	768	12
Large	770M	24	1024	16
3B	3B	24	1024	32
11B	11B	24	1024	128

Table 2.1: Models proposed in the T5 paper in order of increasing complexity.

In Table 2.1 the symbol n_{layers} stays for the number of layers in both the encoder and decoder, d_{layer} is the dimension of the embedding vector, and n_{heads} the number of independent heads in the attention mechanism.

2.4.1 Training strategies

The general approach of training the five models was a single massive pre-training followed by fine-tuning the model for the specific downstream task before being evaluated. In the pre-training, the authors used the C4 dataset [27], a Colossal cleaned version of Common Crawl's web crawl corpus that contains data deriving from web pages.

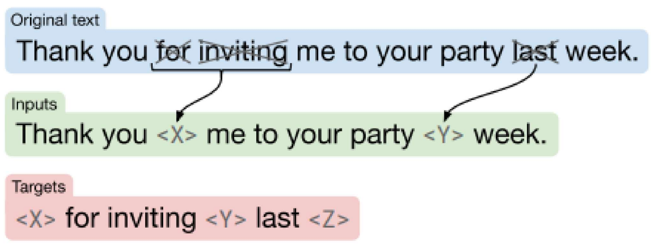
Original text
Thank you ~~for inviting~~ me to your party ~~last~~ week.
Inputs
Thank you <X> me to your party <Y> week.
Targets
<X> for inviting <Y> last <Z>

Figure 2.5: Diagram of T5 text-to-text framework - *source* [21].

Figure 2.5 shows the unsupervised task used in the T5 pre-training which is based on denoising objectives also known as "masked language modeling" introduced by BERT [22]. The model learns to predict and restore missing or corrupted tokens in the input in a denoising objective. In particular, the authors of T5 designed an object that randomly samples and drops out 15% of tokens in the input sentence. Then the selected elements are replaced by sentinel tokens, special tokens added to the T5 vocabulary that must be unique in a single sentence (its ID is never repeated in the sentence). The target consists of all the dropped-out spans of tokens, separated by the same sentinel tokens used in the input sequence, with an additional sentinel token marking the end of the target sequence.
For the purposes of this thesis, particularly relevant is the question-answering fine-tuning that they performed on the SQuAD benchmark [28]. The question-answering involves training models to understand and respond to questions based on the given context, representing the "knowledge" of the model. During the training, the model receives the question along with its context and generates the answer token-by-token. Figure 2.6 shows an example derived from the SQuAD benchmark.

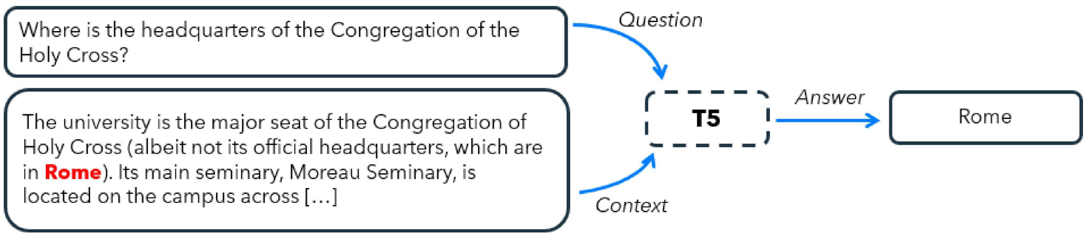
Where is the headquarters of the Congregation of the Holy Cross?
Question
The university is the major seat of the Congregation of Holy Cross (albeit not its official headquarters, which are in Rome). Its main seminary, Moreau Seminary, is located on the campus across [...]
Context
T5
Answer
Rome

Figure 2.6: Example of QA task derived from SQuAD dataset.

2.5 Computer networking

The definition of computer networking is widely discussed due to the complexity of the matter which it represents. A possible definition is the following: *"A computer network is a system of data sources and data receivers that are connected by certain media, transmission, and switching equipment or other networks"* [29].
As we read in the definition, essential is the data transmission between a source and a destination which happens thanks to a structure of protocols that work at different levels of abstraction ruled in the ISO-OSI stack that finds its implementation in the TCP/IP stack.

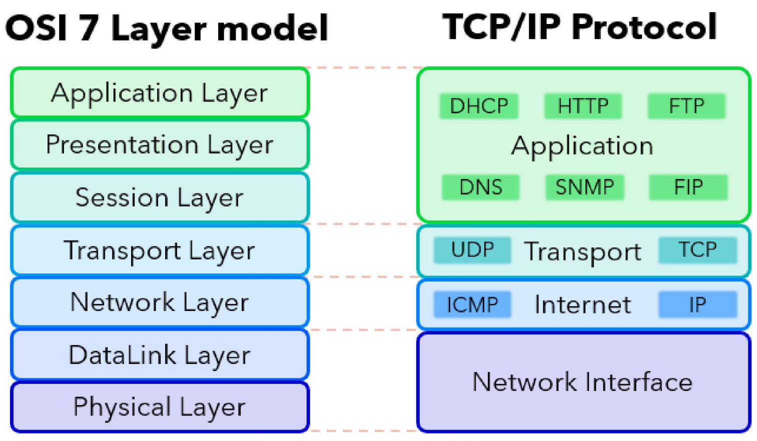

Figure 2.7: The theoretical OSI 7 layers stack and the implemented one TCP/IP.

In brief, the OSI (Open Systems Interconnection) model is a conceptual framework used to understand and implement network communications among different systems. The OSI model divides the communication process into seven layers: Physical, Data Link, Network, Transport, Session, Presentation, and Application. Particularly relevant for the thesis is the Network layer which structures and manages multi-node networks, including addressing, routing, and traffic control.

2.5.1 Internet Protocol (IP)

Internet Protocol is the protocol at the base of networking and makes possible the flow of packets on the internet. It is based on addresses that are unique identifiers assigned to each device connected to a network and they specify where data should be sent and received. There are two main types of IP addresses: IPv4 and IPv6. IPv4 addresses are 32-bit numbers, typically represented in dotted decimal format (e.g., 192.168.0.1), and are running out due to the huge number of devices connected to the internet. To address this problem, IPv6 was introduced, which uses 128-bit

numbers, providing a virtually limitless number of unique addresses.
Figure 2.8 highlights the many differences between the two IP versions. First of all the header length because for IPv6 it is fixed at 60 bytes, while version 4 normally is 20 bytes in length up to 60 bytes if the field "options" is used. Relevant is also the difference in the number of fields. Indeed IPv6 header format is simplified in comparison with IPv4 thanks to the removal of several fields. Relevant to the thesis are the changes introduced with IPv6. Figure 2.8 highlights with different colors the fields' name kept from IPv4 to IPv6 such as the *version field*, the fields with name and position changed, such as *Total Length* (IPv4) called *Payload Length* (IPv6), and finally the fields not kept or added with the new version.

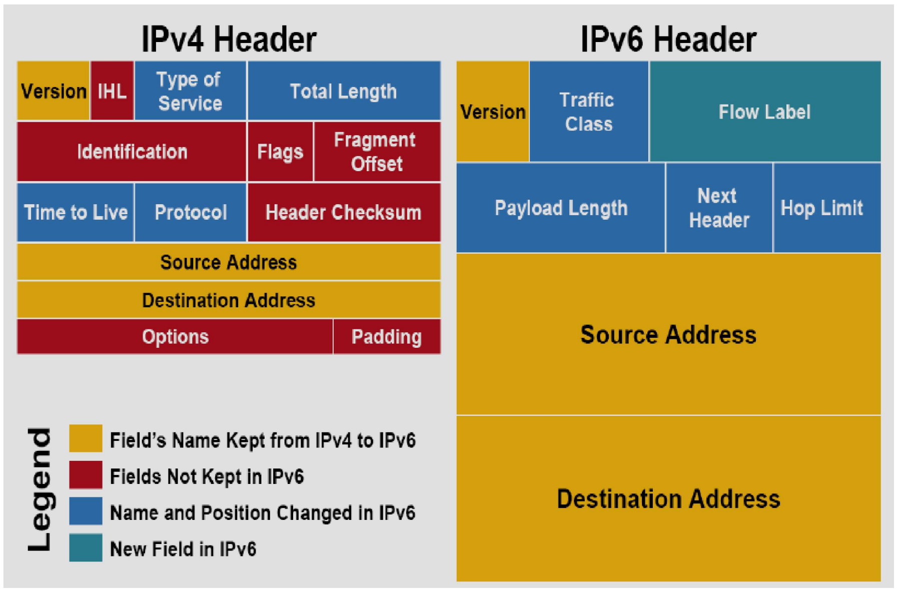

Figure 2.8: Comparison between the IPv4 and IPv6 headers - *source* [30].

2.5.2 Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol is a crucial network layer that is not typically used for exchanging data between systems but for network diagnostics and troubleshooting. Indeed it is essential to send error messages and operational information. It helps diagnose network communication issues by indicating whether data packets have successfully reached their destination or if there were any failures along the communication link. Considering its architecture, ICMP consists of an eight-byte header which follows the normal IPv4 header. As Figure 2.9 shows, it consists of three fields: the type, the code, and the checksum.

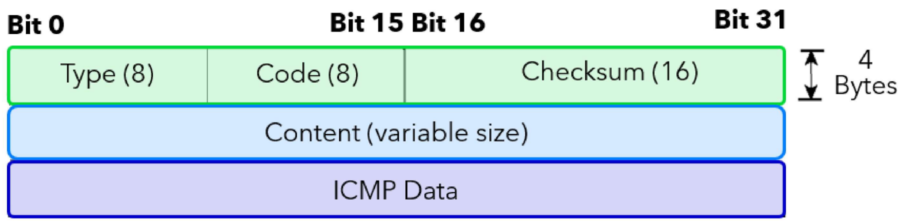

Figure 2.9: ICMP header.

2.5.3 Transmission Control Protocol (TCP)

The Transmission Control Protocol is a fundamental transport layer protocol, designed to provide reliable, ordered, and error-checked delivery of data between applications running on hosts across the internet. TCP is connection-oriented, meaning it establishes a connection through a three-way handshake before data transmission begins. This ensures that data packets are delivered in the correct order and without errors, making TCP ideal for applications where data integrity is crucial, such as web browsing, email, and file transfers. TCP also includes mechanisms for flow control, congestion control, and error recovery, which help maintain the stability and efficiency of data transmission over the network.
To provide all the services listed before, as Figure 3.3 shows, the number of TCP header fields is very high making it 20 bytes long. In addition to the networking fields such as the source and destination ports and the checksum, essential fields are also the acknowledge number, the sequence number, and the window size.

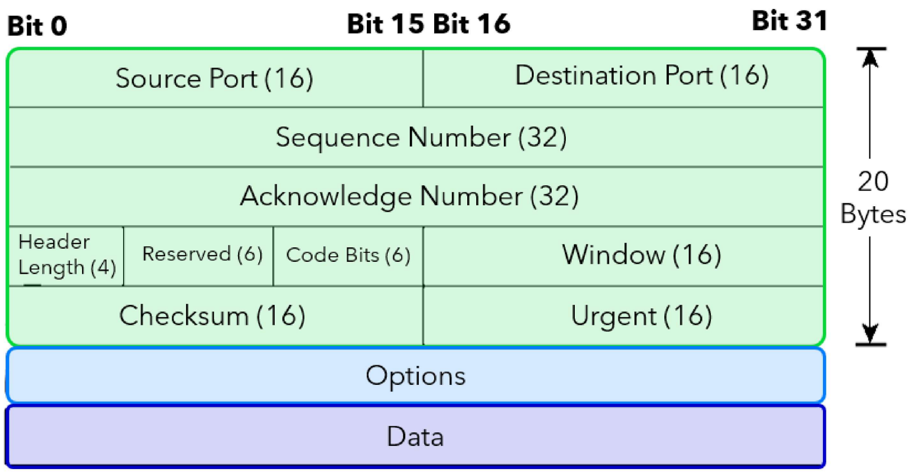

Figure 2.10: TCP header.

2.5.4 User Datagram Protocol (UDP)

The User Datagram Protocol is a fundamental communication protocol belonging to the transport layer in the OSI stack, designed for efficient and fast data transmission. UDP is connectionless, meaning that it sends data without establishing a fixed connection or ensuring the delivery of packets. This makes UDP ideal for applications where speed is crucial and partial data loss is acceptable, such as live video streaming and online gaming. The protocol provides minimal error checking via checksums but does not guarantee packet delivery, order, or protection against duplicates.
Considering the objectives of UDP, the packet header must be simple and it provides the basic networking fields which are source and destination port, the checksum, and the urgent field.

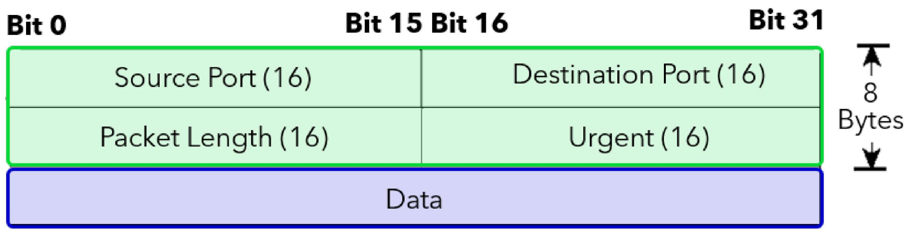

Figure 2.11: UDP header.

Chapter 3

Problem definition and proposed framework

This chapter is about the punctual definition of the problem and the presentation of the structure of the final model implemented. As the following sections will show, the project is divided into two main phases: the first exploits the T5 model to fine-tune its encoder massively. It consists of a denoiser (phase 0) and a question-answering task (phase I). Then, in the second phase, the encoder and the bottleneck associated with a classification header are used to solve different final tasks.

3.1 Problem definition

The objective of the research is to obtain a high-quality packet representation to solve different downstream tasks. The choice of starting from the representation and not from zero introduces modularity in the model, saving training time. In particular, the primary step is to learn an encoder to create a representation of unknown packets starting from the final encoder's hidden state. Afterwards, thanks to the representation is possible to train a classification head to solve downstream tasks, such as application classification, service detection, malware detection, and intrusion detection.
Going into detail, given a packet set $\mathcal{X} = \{x_1, x_2, ..., x_n\}$ where $\boldsymbol{x_i}$ is the i-th packet and $|\mathcal{X}|$ is equal to the total number of packets. Then each packet $\boldsymbol{x_i}$ is represented as an input sequence $\boldsymbol{x_i} = \{t_{i,1}, t_{i,2}, ..., t_{i,n}\}$ where $t_{i,j}$ is the j-th token derived from the split of the tokenizer. The length of the vector $\boldsymbol{x_i}$ can vary since the size of packets is unfixed.
In addition, each packet $\boldsymbol{x_i}$ can assume a number n of labels $\boldsymbol{y_i}$ depending on the number of downstream tasks we want to solve. So, for n downstream tasks, we have $\boldsymbol{y_i} \in \mathbb{R}^n$ for each packet $\boldsymbol{x_i}$. However, for each task, the number of classes is

variable. So, $\boldsymbol{y}_{i,j} = \{1, ..., |\mathcal{C}_j|\}$ where the classification task j has class set $\mathcal{C}_j$ and the number of classes in this task is $|\mathcal{C}_j|$.
Moreover, to pass from the packet to its classification, we need the latent vector (packet representation) $\boldsymbol{r_i} \in \mathbb{R}^d$ where d is the hidden dimension of the model. The latent vector is obtained by combining the columns of the latent matrix $\boldsymbol{H_i} \in \mathbb{R}^{d \times L}$ where L is the number of tokens in each packet and d the dimension of a token e_i. Following the previous notations, the formalization of the objective of packet representation learning is:

"*Given the input set $\mathcal{X}$ and the label set $\mathcal{Y}$ of multiple classification tasks, the goal is to:* (1) *learn a packet representation encoder $f : \boldsymbol{x_i} \rightarrow \boldsymbol{H_i}, \boldsymbol{r_i}$;* (2) *obtain accurate $\boldsymbol{y_i}$ on downstream tasks by a function $g : \boldsymbol{H_i}, \boldsymbol{r_i} \rightarrow \boldsymbol{y_i}$".* [4]

3.2 Phase I: encoder training

The core part of the thesis is the pre-training executed in this phase. The base architecture is the one proposed by Google with T5 [21] already explained in section 2.4. However, as Figure 3.2 shows, I have introduced a layer that transforms the hidden state $\boldsymbol{H_i}$ into the packet representation $\boldsymbol{r_i}$. Essentially, phase I consists of three main key elements: the initial encoder, the bottleneck, and the decoder.

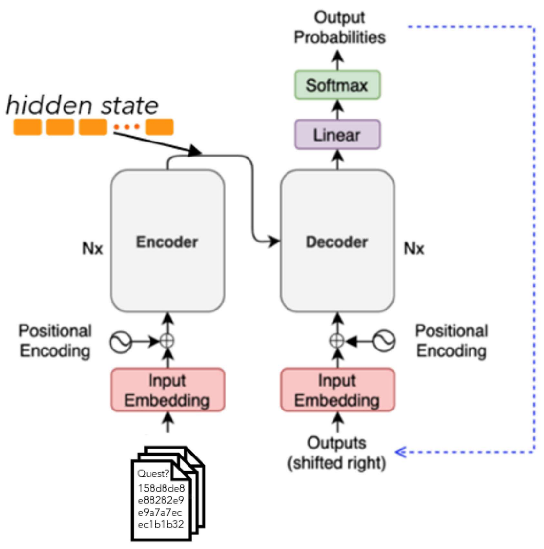

Figure 3.1: Schema of the architecture of the T5 model.

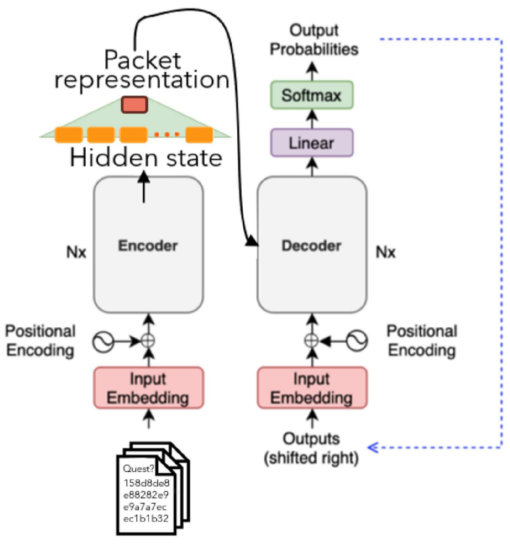

Figure 3.2: Schema of the modified structure of the T5 model for our objectives.

3.2.1 Initial encoder

Before entering the encoder, the encrypted and decrypted packets are represented by a string of hexadecimal numbers. Since I used a T5 pre-trained model, a question is brought forward at the beginning of the string to ask for a specific field of the packet. Afterwards, thanks to the tokenizer and the *input embedding* layer, for each token $t_{i,j}$, a vector of embeddings $\boldsymbol{e_{i,j}}$ of size $d = 768$ dimensions, is obtained. By passing through the laters of the encoder, the embeddings are modified and in the end, we get:

$$\boldsymbol{H_i} = Encoder(\boldsymbol{x_i})$$

where $\boldsymbol{H_i}$ contains the embeddings vectors $\boldsymbol{e_{i,j}}$ for each token giving a compact representation of the input text. For this reason $\boldsymbol{H_i} \in \mathbb{R}^{d \times L}$; for each packet, there are maximum L tokens, each one encoded into d dimensions.

3.2.2 Bottleneck

The bottleneck is one of the key components of the research because it tries to obtain a single representation for the entire packet from the L representations of the tokens. The main problem to be solved is keeping as much of the original information inside each packet as possible. Indeed, the dimensionality reduction process necessarily discards a part of the information set.
In literature, many possibilities are present starting from the easiest to more complex solutions. In particular, I have experienced four solutions:

1. **First pooling**: a dummy solution that takes as packet representation the embedding of the first token that is always the initial part of the question needed by T5. The representation vector of packet i becomes:

$$\boldsymbol{r_i} = \boldsymbol{e_{i,0}}$$

2. **Mean pooling**: performs the average over the hidden vectors $\boldsymbol{e_{i,j}}$ of the hidden matrix. The representation vector of packet i becomes:

$$\boldsymbol{r_i} = \frac{\sum_{j=0}^{L} \boldsymbol{e_{i,j}}}{L}$$

3. **Luong attention** [31]: performs a weighted average of the embeddings. The weights, computed for each $e_{i,j}$, must be positive and the sum is 1. The representation vector of one packet becomes:

$$w_j = \frac{\exp{(\boldsymbol{e_j}^\top \boldsymbol{q})}}{\sum_{z=0}^{L} \exp{(\boldsymbol{e_z}^\top \boldsymbol{q})}} \qquad \boldsymbol{r} = \sum_{j=0}^{L} w_j \boldsymbol{e_j}$$

where $\boldsymbol{q}$ is a learnable query vector and w_j is the weight associated with the embedding vector $\boldsymbol{e_j}$.

4. **No bottleneck**: this solution doesn't create a packet representation, but, as Figure 3.1 shows, $\boldsymbol{H_i}$ doesn't change. This implementation is provided as a sanity check of the program.

3.2.3 Decoder

As reported in subsection , the decoder takes two elements as input:

1. **Contextualized encoder representation**: the output of the encoder part which is equal to $\boldsymbol{H_i}$ if the bottleneck is not present, $\boldsymbol{r_i}$ if present. This element contains all the information on the question and the packet that the decoder can use to generate the answer.

2. **Answer**: the target that the decoder should generate. Before entering the decoder it is processed as the encoder input. The answer is tokenized and then translated into embedding vectors thanks to the *input embedding* layer. In addition, the sequence of embeddings is right-shifted by one to have the uni-directionality of the decoder that can only see tokens in the past and not in the future.

The decoder is based on a loss function that computes the difference between the token index in input and the one generated by the model. I have followed the formalization provided by the library of python, torch:

$$l(x, y) = L = \{l_1, ..., l_N\}^\top \qquad l_n = -w_{y_n}\, log\, \frac{\exp\left(x_{n,y_n}\right)}{\sum_{c=1}^{C} \exp\left(x_{n,c}\right)} \cdot 1\{y_n \neq -100\}$$

where the indices are in the range $[0, C)$, x the input, y the target, w the weight, C the number of classes, N spans the batch dimension, and then the mean is performed:

$$l(x, y) = \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n} \cdot 1\{y_n \neq \text{ignore index}\}}\, l_n$$

The final goal is to minimize the loss function.

3.3 Phase 0: the denoiser

Ahead of phase I, the encoder-decoder model could be pre-trained on a denoising task already introduced in subsection 2.3. The base architecture exploited is the same introduced in phase I that Figure 3.2 shows. However, the objective of the training changes. The model must now reconstruct a clean "repaired" input from a corrupted, partially destroyed one. This is achievable by firstly corrupting the initial input $\boldsymbol{x}$ to get a partially destroyed version $\tilde{\boldsymbol{x}}$ employing a stochastic mapping function $\tilde{\boldsymbol{x}} \sim q_D(\tilde{\boldsymbol{x}}|\boldsymbol{x})$. The corrupting process is based on the corruption ratio η that defines the number of tokens corrupted inside the input. In particular, considering the number of tokens of the input string ($|\boldsymbol{x}|$), the fixed number η_d is derived as follows:

$$\eta_d = (|\boldsymbol{x}| - |\boldsymbol{x}_{question}|) \cdot \eta$$

where $|\boldsymbol{x}_{question}|$ is the number of tokens that correspond to the T5 question.
Then a η_d number of tokens are chosen at random among the ones not corresponding to the question, and their value randomly changed considering the entire vocabulary indexes, the others are left untouched. All information about the chosen components is thus removed from that particular input pattern, and the autoencoder will be trained to restore them. The corrupted input $\tilde{\boldsymbol{x}}$ is then mapped, as with the basic autoencoder, to a hidden representation $\boldsymbol{r} = f(\tilde{\boldsymbol{x}})$ from which $\boldsymbol{z}$ is reconstructed as $\boldsymbol{z} = g(\boldsymbol{r})$. The parameters are trained to minimize the loss function already explained in subsection 3.2.3.

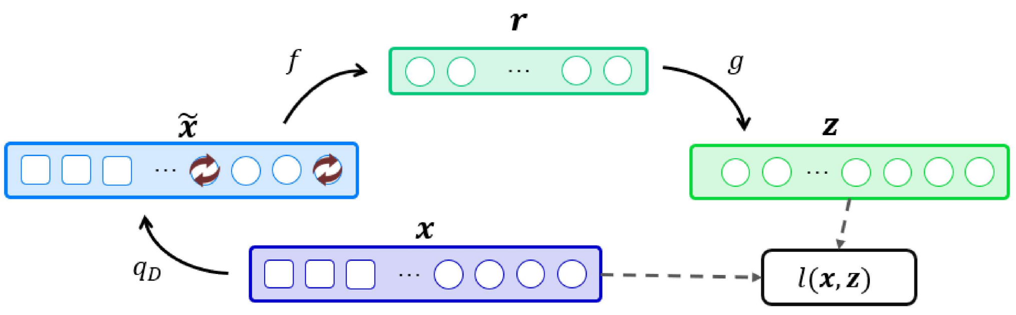

Figure 3.3: Example of denoiser where $\boldsymbol{x}$ is corrupted to $\tilde{\boldsymbol{x}}$, then $\boldsymbol{r}$ is obtained and $\boldsymbol{z}$ is the attempt to reconstruct $\boldsymbol{x}$. The performance is evaluated through the loss function $l(\boldsymbol{x}, \boldsymbol{z})$.

3.4 Phase II: classification

The addition of the classification head serves as a crucial component in the pipeline, enabling the model to perform specific classification tasks with high accuracy. As reported in Section 3.1, we train a model to transform the initial packet representation to another, specifically refined for the downstream task.

$$g : \boldsymbol{H}_i, \boldsymbol{r}_i \rightarrow \boldsymbol{y}_i$$

By leveraging the rich, encoded representations generated by the encoder and bottleneck $\boldsymbol{r}_i$, the classification head can make informed decisions based on it, obtaining the refined version $\boldsymbol{y}_i$. This integration not only streamlines the process

of feature extraction and classification but also demonstrates the versatility and adaptability of the fine-tuned model across different tasks.
The complexity of the classification head was intentionally kept low, based on the idea that the packet representation from the bottleneck requires only slight refinement. For this reason, we introduced a 2-linear layer classification head balancing between model complexity and computational efficiency.
Going into depth on the architecture, the first linear layer takes the encoded representations from the bottleneck as input. This layer applies a linear transformation to the input data, basing the transformation on a set of weights and biases that are learned during the training process. This layer is followed by an activation function, typically a ReLU (Rectified Linear Unit), which introduces non-linearity into the model, allowing it to capture more complex patterns in the data. The second linear layer further processes the output from the first layer, refining the feature space even further. The output of this layer is then passed through a softmax function, which converts the raw scores into probabilities, providing a clear and interpretable classification result.

Chapter 4

Datasets and data processing

This chapter discusses the data used throughout the research, considering the various components of the developed model. The first part presents each dataset used during phases I and II, highlighting their content and the rationale behind their use. The key is that the data sets used in the two phases must differ. The following part describes the pipeline implemented to transform the raw data into the input given to the model.

4.1 Datasets: phase I

As widely explained in section 3, the goal of phase I is to fine-tune the T5 model to enable it to handle raw packages. In this phase, I used two well-known public datasets [32] [33], one coming from a blog that archives malware [34], and finally two generated by passively collecting traffic under the Polytechnic of Turin network and a domestic LAN. All datasets consist of packet-level information stored in PCAP, files used specifically to store network traffic data.

4.1.1 MAWI

The MAWI (Measurement and Analysis on the WIDE Internet) [33] dataset is a comprehensive collection of network traffic data developed inside the WIDE project, a research consortium in Japan established in 1987. Its goal is to promote traffic analysis research and the development of tools. In particular, it has been projected to evaluate the automated anomaly detection methods.
The archive has been operative since 1999 collecting a huge amount of data. In order to obtain data with good quality they are collected on the Japan backbone network (Fig. 4.1) that connects WIDE (Widely Integrated Distributed Environment) to the upstream internet service provider.

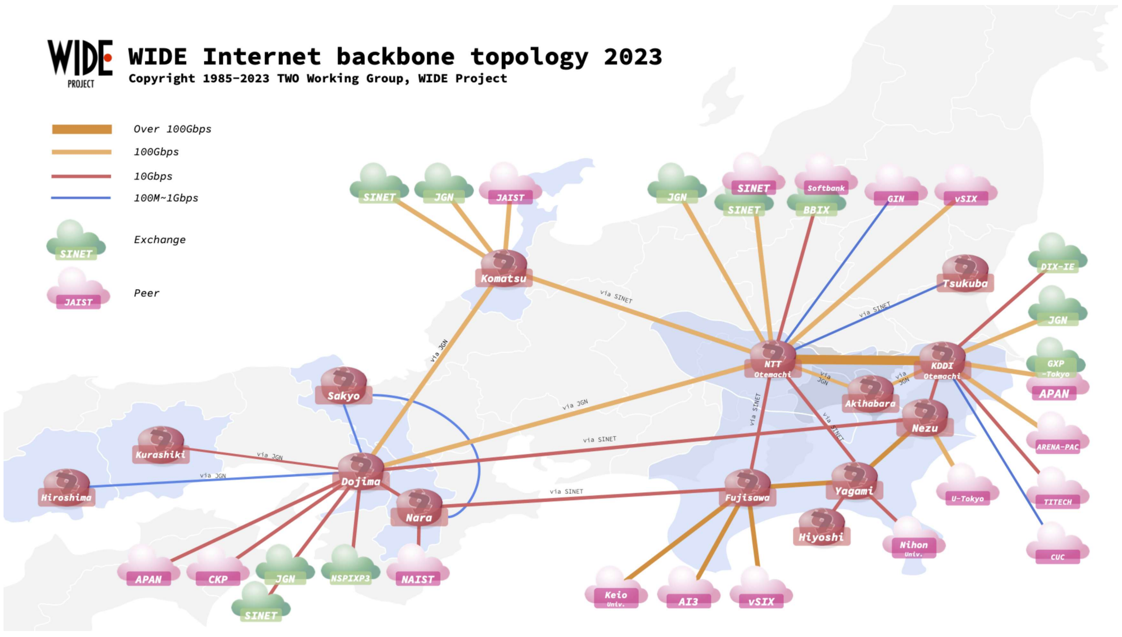

Figure 4.1: Structure of the WIDE infrastructure - *source* [35].

The network topology and the location of the sampling point enable realistic traffic, including academic and commercial traffic, to be collected at a high bit rate.
As Fig. 4.2 shows, the type of traffic is broad and reflects changes in traffic over the years. For example, Fig. 4.2 reveals some of the most prevalent malware in specific years, such as *Code Red*, *Blaster*, and *Sasser*, worms that sadly disrupted Internet traffic worldwide from 2001 to 2006 [36]. And, some choices related to academic research, such as the surge in Teredo traffic in 2010, caused by IPv6 traffic being temporarily tunneled by the *Tokyo6to4* project.

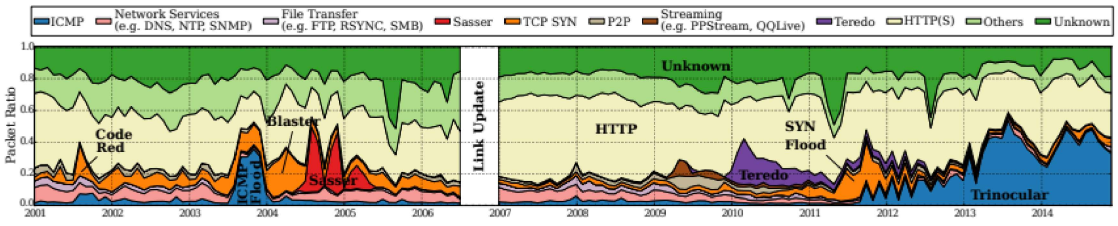

Figure 4.2: Analysis of MAWI traffic in the period 2001/2015 - *source* [36].

Due to the presence of raw data, the MAWI group has outlined a guideline [37] to protect user privacy, removing sensible information. The guideline consists of two principal rules:

- **Payload removal**: in TCP and UDP packets the payload is removed if it contains private information. Also, the header must be removed as a precaution if it might contain sensitive information.

- **IP address scrambling**: each user and organization IP is mapped to another IP address via a hash function. Exceptional, broadcast, multicast, and private addresses may not be scrambled.

From this huge archive, which contains captures from 1999 to the present, I have selected an infinitesimal part consisting of the dump on Sunday, April 2, 2023, lasting 15 minutes (between 14:00:00 and 14:15:00).

4.1.2 UNSW-NB15

The UNSW-NB15 dataset [38], developed by the University of New South Wales in 2015, is a comprehensive resource designed to evaluate network intrusion detection systems (NIDS). The dataset is widely used in cybersecurity research to develop and test new methods for detecting and mitigating network intrusions. Indeed, it labels nine types of attacks: fuzzers, analysis, backdoors, denial of service (DoS), exploits, generic, reconnaissance, shellcode, and worms.
It was created using the IXIA PerfectStorm tool to generate a mix of real normal activities and synthetic attack behaviors, capturing 100 GB of raw network traffic. As Figure 4.3 shows, they used three virtual servers, servers 1 and 3, to spread normal traffic, while server 2 had malicious activities. The data acquisition is performed on router 1 through the tcpdump tool, which acquires both traffic generated by the IXIA tool and normal traffic.

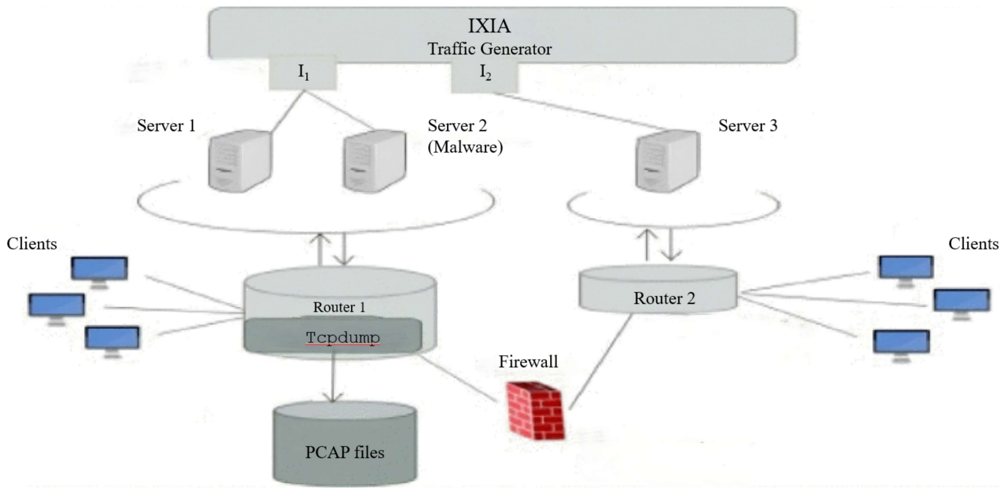

Figure 4.3: Testbed for the UNSW-15 dataset - *modified from* [38].

The dataset consists of four CSV files, each corresponding to a distinct category of attacks. Additionally, it includes access to the raw PCAP files, which contain the original packet-level data captured during the collection process. The packets are not anonymized and present the full payload, however, due to the acquisition testbed doesn't have the ethernet layer. Indeed, the generation of the packet's link-layer header is too heavy for tcpdump, so it constructs a synthetic one.
For my objectives, I have selected only a small PCAP file from January 22, 2015, of size around 50 MB

4.1.3 Passively collected PCAP

To increase the variety of data inside the final dataset, I have introduced two datasets collected through the packet analyzer Wireshark on a VivoBook ASUS Laptop X509DA. The created dumps are two:

- **Domestic LAN**: it was executed in the afternoon of July 18, 2024, in my home. During the acquisition, different types of applications and services were used to have a wider type of traffic. In particular, the highest amount of data consists of a big file transfer from my PC to the SmartData cluster. Moreover, I visualized a short video on YouTube, sent some emails with Outlook and Gmail, and finally opened some content on Google Drive. The amount of traffic is 100 MB containing mainly TCP and UDP traffic.

- **Polytechnic of Turin network**: it was performed during the working day of 24 July 2024 in the SmartData department. During the acquisition, in parallel to the normal traffic of the department, I opened a YouTube video and different academic websites in the background, and then, I searched some paths via Google Maps and scrolled through my main LinkedIn page. The resulting traffic was around one gigabyte and a half of data, in the majority of types TCP and UDP (QUIC).

4.1.4 PCAP from malware-traffic-analysis blog

Malware-traffic-analysis [34] is a blog containing many Windows-based malware samples as a resource for threat researchers and other security professionals. The dumps are archived by year and in each PCAP file is present the starting date in which the malware has been collected.
In order to have recent data and malicious traffic, I have selected two PCAPs from this blog. The first corresponds to a *Latrodectus infection* that occurred on 2024-06-25, while the second contains a *WikiLoader infection* that occurred on 2024-01-17. Considering the two files the total amount of data is around 50 MB

4.2 Datasets: phase II

Compared to Phase I, the objective and constraints have changed a lot. In phase II, I need a set of datasets that are different from the ones used before and that are labeled. Indeed, the model must classify packets the encoder has not yet seen to avoid overfitting problems.
After carefully analyzing already implemented solutions in the literature, I found NetBench [39] in which the authors proposed a well-structured benchmark.

4.2.1 NetBench

Existing studies often use data processing techniques specifically designed for their own purposes. Furthermore, some research randomly divides packets from the same flow into training and testing sets for packet-level evaluation, which can result in data leakage due to the strong correlation between packets from the same flow. For these reasons, the authors proposed NetBench [39], an extensive and detailed benchmark for network traffic analysis, encompassing 7 datasets, used to obtain 15 classification tasks along with 5 generation tasks. Here is the list:

- **ISCXVPN 2016** [5]: The dataset, created by the Canadian Institute for Cybersecurity, is a comprehensive collection of network traffic data designed to represent real-world traffic scenarios. It includes VPN and non-VPN traffic, captured from various applications, such as Skype, email clients, and Facebook, and various services, including VoIP, P2P, and streaming.

- **ISCXTor 2016** [40]: Also this dataset was developed by the Canadian Institute for Cybersecurity and is a comprehensive collection of network traffic data including both Tor and non-Tor traffic. The dataset includes labels for web browsing, email, chat, audio and video streaming, FTP, VoIP, and P2P services.

- **USTC-TFC 2016** [41]: The dataset includes benign and malicious traffic samples, with benign traffic covering applications like BitTorrent, Skype, and Gmail, and malicious traffic featuring various malware such as Cridex, Geodo, and Zeus

- **Cross-Platform Android and iOS** [42]: The dataset was designed for mobile application development and security. It includes data collected from both Android and iOS platforms, covering various aspects such as app usage, network traffic, and user interactions.

- **CIRA-CIC-DoHBrw2020** [43]: The dataset, developed by the Canadian Institute for Cybersecurity, is designed to analyze and evaluate DNS over

HTTPS (DoH) traffic. It includes both benign and malicious DoH traffic, as well as non-DoH traffic, captured using various browsers and DNS tunneling tools.

- **CIC IoT Dataset 2023** [44]: The dataset, developed by the Canadian Institute for Cybersecurity was designed to support research in IoT security by providing a realistic representation of IoT network traffic. It includes data from 105 IoT devices and covers 33 different types of attacks, categorized into seven groups: DDoS, DoS, Reconnaissance, Web-based, Brute Force, Spoofing, and Mirai.

Since the authors' goal was to obtain standardization of the datasets, they further processed them. As the first step, they divided each one into training, validation, and testing sets to avoid high-correlated packets from the same flow being present in both training and testing data. Then, to protect data privacy, they anonymized the packets by replacing source/destination IP addresses and port numbers with 0. The second step consists of obtaining a unified representation of the packet. They decided to convert the flows into a hexadecimal format, segment data into 4-digit blocks, and incorporate specific symbols to identify the header and the payload. To provide a complete benchmark they further provide a flow representation.

Restricted benchmark

Considering that the thesis has an explorative focus, only a subsample of Netbench was considered. In particular, I selected five tasks by choosing both the easiest and the most difficult ones to test my model in various scenarios. Table 4.1 reports the datasets and tasks considered.

Dataset	# Packets	N° task	# Labels	Labels
ISCXVPN2016	Train: 355 033 Val: 43 957 Test: 45 006	1	Binary	VPN, non-VPN
		2	Multiclass (6)	Service Detection: *P2P, FileTransfer, VoIP, etc.*
		3	Multiclass (16)	Application Detection: *Gmail, Spotify, Vimeo, FTPS, etc.*
USTCTFC2016	Train: 1 537 751 Val: 193 927 Test: 190 344	6	Binary	Malware, non-Malware
		7	Multiclass (19)	Application Detection: *Htbot, Skype, Tinba, MySQL, etc.*

Table 4.1: Benchmark used to evaluate the performance of the proposed model.

Among the five tasks illustrated in Table 4.1, I have selected the number 2 as the *reference* to obtain the first results.

4.3 Data elaboration: from PCAP to model input data (phase I)

Starting from the PCAP files it is necessary to process the data to make it readable by the model. As the pipeline in Figure 4.4 shows, after the download of the PCAP from the internet, the file is, at first, preprocessed in Wireshark for raw filtering, and then finely processed through the python library `scapy` [45]. The final output is saved on a CSV file.

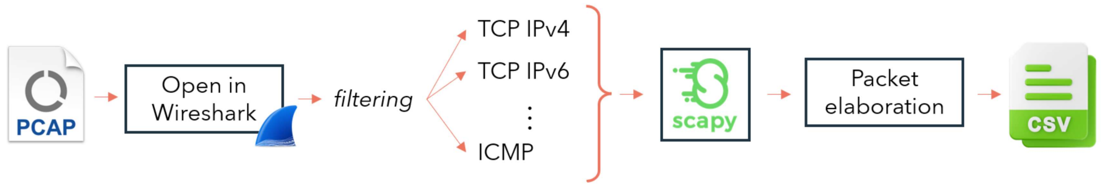

Figure 4.4: Pipeline for data processing.

4.3.1 Wireshark raw filtering

As a first step, the PCAP files were downloaded on my PC VivoBook ASUS X509DA and opened with the Wireshark distribution for Windows systems. Thanks to the power of this application a brief analysis was performed identifying the type of traffic present and the packet length. Figure 4.5 shows some of the most present protocols of levels 3 and 4, while 4.6 reports the CDF (cumulative distribution function) of the number of packets and bytes referred to the packet's length.

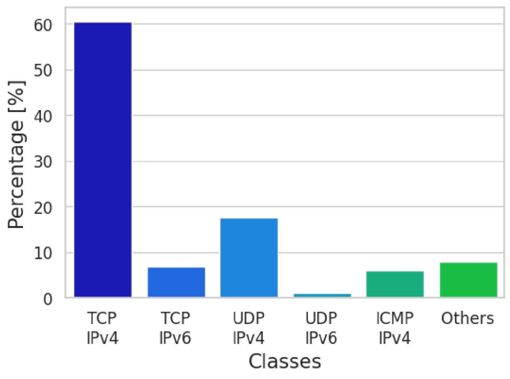

Figure 4.5: Protocol hierarchy statistics (MAWI - April 2, 2023).

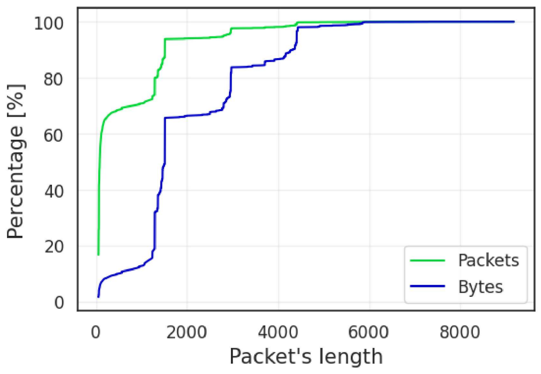

Figure 4.6: CDF of packet length and byte amount (MAWI - April 2, 2023).

Afterward, depending on the statistics previously analyzed, I created different PCAP files. The possibilities were at the network layer, IPv4 and IPv6, while at the transport layer, TCP, UDP, and ICMP. The process is easily performed using the filters provided by Wireshark. For instance, to select the TCP packets on IPv4, the filter to use is:

```
ip.version == 4 && tcp
```

Since the elaboration through `scapy` is time-consuming, only the first N packets are saved to speed up the process, where N is the number of packets necessary for the dataset.
At the end of this process, different sub-dumps are created from the original PCAP file, each containing a type of traffic only (TCP-IPv4, TCP-IPv6, UDP-IPv4, etc.).

4.3.2 Scapy fine process

Fundamental in this step is the python library `scapy`. It is a powerful tool for packet manipulation that can create or decode packets across various protocols, send and capture them, match requests with responses, and other useful features. The library handles common tasks such as scanning, tracerouting, probing, unit testing, attacks, and network discovery [45].
For my purposes, I used a tiny part of the library's functionalities, by reading the PCAP files, packet by packet, and capturing the necessary fields. In particular, the following pseudo-code was applied:

Algorithm 1 Packet processing algorithm - part 1

```
    Input: PCAP_file
    Output: table df_packets with processed packets
 1: for packet in PCAP_file do
 2:     Randomly change IP of packet;
 3:     Randomly change TTL of packet;
 4:     Remove the Ethernet header of packet;
 5:     Check and compute the checksum of packet;              ▷ On IP layer
 6:     Compute the last byte and length of packet payload;    ▷ On IP layer
 7:     Generate a dictionary of packet;         ▷ With fields in hexadecimal
 8:     Generate the hexadecimal string of packet;
 9: end for
10: Save all in df_packets;
```

By looking at Algorithm 1, it is possible to see that seven steps are performed on each packet. Let's analyze briefly the reasons behind each one considering that the goal (reported in Section 3.2) is to train a model that generates a significant representation of each packet. Here are the steps' comments:

- **Steps 2 and 3**: the TTL (time to live), the source, and the destination IP are randomly changed to reduce the probability of overfitting during training. Indeed, the PCAP files contain sets of conversational flows that always have the same IPs. So the model could memorize these IPs without learning where to find the information. The same applies to the TTL, which usually assumes around 20 values instead of the permitted 255.
- **Step 4**: the ethernet layer is deleted because it does not give useful information on the packet. Indeed, the physical layer has meaning only on the LAN.
- **Steps 5 and 6**: the idea is to generate some fields that are not just retrieval, but imply reasoning for the model. In addition, since the source and destination IP of the packet are changed, the checksum must be corrected accordingly.
- **Steps 7 and 8**: for each packet two representation are derived. The first is a dictionary with all the packet's fields reported in hexadecimal format. The second is a string with the bytes of the packet separated by a white space.

As reported in Section 2.4, since the T5 model has a question-answering structure, I needed to transform my dataset to have some questions related to its fields for each packet. For this reason, the simple Algorithm 2 was applied.

Algorithm 2 Packet processing algorithm - part 2

```
    Input: df_packets, and $P_{fields}$ that is the set of all packet's fields askable.
    Output: QA_dataset
1: $count = 0$
2: for each row in df_packets do
3:     Select randomly a number $n_{questions}$        ▷ where $n_{questions} \leq n(P_{fields})$
4:     For each question generates a row in QA_dataset
5:     if $count \geq MAX_{questions}$ then
6:         End the loop
7:     end if
8:     $count \leftarrow count + n_{questions}$
9: end for
```

Thanks to Algorithms 1 and 2 it is possible to create a wide range of different datasets that can vary in complexity, by selecting easy or hard questions only, or in length by changing the parameter $MAX_{questions}$.
To have an idea of the possible questions that can be found in the dataset, Table 4.2 shows them for the case TCP over IPv4. In the other scenarios (UDP and

ICMP over IPv4/IPv6) the questions are a sub-sample to consider the difference in the protocol structure.

IP *source*	IP *id*	TCP *checksum*	TCP *sequence number*
IP *destination*	IP *checksum is correct?*	TCP *window*	TCP *payload length*
IP *ttl*	TCP *source*	TCP *ack*	TCP *last byte header*

Table 4.2: Possible questions performed on packets of type TCP over IPv4.

Considering the other scenarios (UDP and ICMP over IPv4/IPv6) the questions are a sub-sample of the fields in Table 4.2. Indeed, it is necessary to consider the difference in the architecture of the protocols. In the following, I provide examples of T5 prompts:

- *What is the TCP checksum?*`<sep>`4500 0028 8145 4000 ... CD19
- *What is the destination IP of the packet?*`<sep>`4500 05D4 741D 4000 ... F7C6
- *Is the packet's IP checksum correct?*`<sep>`4500 00AA D973 4000 ... 08D4

Task complexity datasets: Easy, Hard, and Hybrid

Thanks to the data processing explained, I created three datasets to highlight the model's performance under different scenarios using the data, of datasets described in Section 4.1, processed as explained previously. Table 4.3 shows the dataset composition highlighting its number of samples, the asked fields, and the type of traffic.

	N° questions	**Question type**	**Traffic type**
Easy	50 000	IPttl, srcIP, dstIP, IPid, TCPseq, srcTCP, TCPchk, TCPack, TCPwnd	TCP-IPv4
Hard	50 000	IPttl, srcIP, IPid, dstIP, last_header3L_byte, len_payload, checksum_check	TCP-IPv4, TCP-IPv6, UDP-IPv4, ICMP
Hybrid	50 000	IPttl, srcIP, dstIP, IPid, TCPseq, srcTCP, TCPchk, TCPack, TCPwnd last_header3L_byte, len_payload, checksum_check	TCP-IPv4, TCP-IPv6, UDP-IPv4, ICMP

Table 4.3: Description of the datasets Easy, Hard, and Hybrid.

The goal of the *Easy* one is to implement a retrieval model that given a specific question, can understand it and find the answer in the context. On the other hand, the *Hard* dataset tries to force the model to perform more complex tasks, such as the computation of the checksum, or the payload length. The *Hybrid* is just the mix of 50% of *Easy* and 50% of *Hard*. Therefore, the *Hybrid* dataset contains the same type of traffic as *Hard* but the presence of more questions related to TCP-IPv4 makes this protocol prevalent to the others.

Chapter 5

Evaluation and discussion of the results

In this chapter, I show the entire set of experiments, and the corresponding results, performed during the explorative phase. Indeed, the final pipeline, explained in the previous chapters, resulted from many experiments and wise choices taken in order to satisfy the objectives of the thesis.
It is possible to divide this chapter into two parts, the first is the explanation of the experiments performed in each phase, while the second shows the final results of the downstream tasks considering the scenario in which I assume the best parameters.

5.1 Systems for experiments: HPC and BigData

To initiate my coding work, I started with the one provided by the LogPrécis repository [46, 47]. This gave me a robust starting point ensuring a more efficient and streamlined development process.
The experiments were conducted on two distinct clusters. All training models were executed on the Polytechnic HPC cluster [48], which comprises 57 nodes and 1824 computing cores. Of these, 6 nodes were primarily utilized, each equipped with 4 Nvidia Tesla V100 SXM2 GPUs, featuring 32 GB of VRAM and 5120 CUDA cores per GPU, totaling 24 GPUs. The system operates on CentOS 7.6 with SLURM 18.08.8 as the job scheduler.
Additionally, all debugging operations were performed on the BigData@Polito cluster [49], which consists of over 36 nodes. In this setup, only 2 nodes are GPU-enabled, each housing 2 Nvidia Tesla V100 GPUs with 16 GB of VRAM, amounting to 4 GPUs in total.

5.2 Experiments

To obtain a good packet representation it is necessary to perform a series of experiments designed to optimize the choice of each parameter and understand how each one affects the final result. The majority of these experiments typically consist of a grid search focusing on the most likely areas of the solution space. Indeed, greedy choices are needed due to the impossibility of exploring the entire solutions space.
Before the illustration of the experiments performed, it is important to underline that:

1. In Phase 0 and Phase I, the datasets are those detailed in Section 4.3.2. As previously noted, these datasets are entirely distinct from those employed during the classification phase.

2. In Phase II, the results are compared on *Task2*, as already mentioned in Section 4.2.1, because the task is reasonably challenging.
   Task2 consists of classifying packets into six service classes. the dataset is extremely unbalanced so Table 5.1 reports the classes and the amount of data for each.

	Chat	**Email**	**FileTransfer**	**P2P**	**Streaming**	**VoIP**
Train	22 540	17 712	94 970	8 171	18 556	193 084
Validation	2 588	2 140	11 729	1 189	1 813	24 498
Test	2 899	2 526	11 952	916	2 331	24 376

Table 5.1: The compositions of the training, validation, and test sets for the classification task, referred to as *Task2*.

5.2.1 Choice of initial parameters

During the initial experiments, I made several arbitrary choices, which were subsequently modified or confirmed during the exploration phase. The following list details these choices and provides the primary reasons behind them.

- **Dataset choice:** As explained in subsection 4.3.2, I generated three different datasets in growing order of complexity: *Easy*, *Hybrid*, and *Hard*. The initial choice is to use the *Hybrid* one because it includes the other two datasets.

- **Format choice:** I formatted the model's input string, maintaining the Wireshark style called *every2* in the following experiments. It consists of a string of hexadecimal numbers each separated by a white space.

- **Denoiser**: in the early stages, the denoiser is not applied. So, in Phase I, the encoder is directly loaded from the T5 library. The choice was performed not to overlap the effects of the two pre-training tasks (Phase 0 and Phase I).

With each successive experiment, the optimal parameters identified are incorporated into subsequent experiments.

5.2.2 Effect of bottleneck choice

The core choice of the thesis was to determine the optimal representation of the input packet by compressing the information from the encoder's last hidden state. As detailed in Section 3.2.2, the alternatives analyzed include first, mean, and Luong attention mechanisms. The experiment aimed to identify the most suitable learning rate for each method and subsequently compare their performance on the same test set. Both the training and test sets were derived from the *Hybrid* dataset, as explained in Subsection 5.2.1. Initially, the best learning rate for each bottleneck was selected, followed by a comparison of the best results for each method.
Table 5.2 shows the results in terms of accuracy.

	First	**Mean**	**Luong attention**
LR $= 1 \cdot 10^{-4}$	93.01	94.94	94.59
LR $= 5 \cdot 10^{-4}$	**96.94**	**97.62**	**98.12**
LR $= 1 \cdot 10^{-3}$	93.97	97.06	35.61

Table 5.2: Accuracy [%] on validation for bottleneck with different learning rates (LR) with *Hybrid* dataset.

Examining Table 5.2, it is evident that all three solutions perform optimally with a learning rate of $5 \cdot 10^{-4}$. The *Luong attention* bottleneck achieves the highest accuracy, closely followed by the *mean* bottleneck. Additionally, when analyzing the models' behavior with varying learning rates, I observed that the *mean* and *first* bottlenecks exhibit greater stability, with only about a 3% drop in accuracy. In contrast, the *Luong attention* bottleneck's performance degrades significantly at higher learning rates, such as $1 \cdot 10^{-3}$, where its accuracy drops to around 35%.
Following this analysis, I selected a learning rate of $5 \cdot 10^{-4}$ for all bottlenecks and evaluated their performance on a common test set using the *Hybrid* dataset. The results in Table 5.3 align with the trends observed during validation.
The last evaluation assessed the extent to which the bottleneck degrades the model's performance by analyzing both accuracy and convergence speed during training. In particular, I compared the three bottlenecks with the case *none*, in which the

	Accuracy on test
First	93.69
Mean	93.85
Luong attention	94.36

Table 5.3: Accuracy [%] on test for bottleneck with *Hybrid* dataset.

model can use the representations of each sentence token and not only a compressed version. As I expected, Figures 5.1 and 5.2 show the *none* case is always the best in terms of accuracy and convergence time. Indeed, the loss immediately starts at 0.1, while the others assume value around 1 or above, and after a few epochs, it achieves 0.01. Similar considerations can be made regarding the accuracy. Indeed *none* achieves the best score (99.4%) and already at the first epochs, its accuracy is around 95%.
Therefore, after all the previous considerations, I can conclude that introducing a bottleneck in the model degraded the performance. However, it is essential for the thesis since I need a single embedding representing the entire packet.

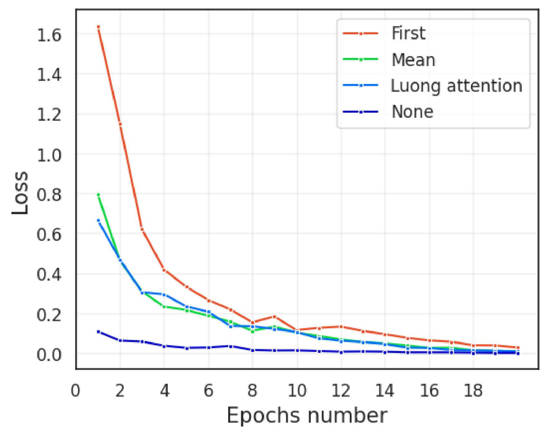

Figure 5.1: Loss curve of the four bottlenecks analyzed.

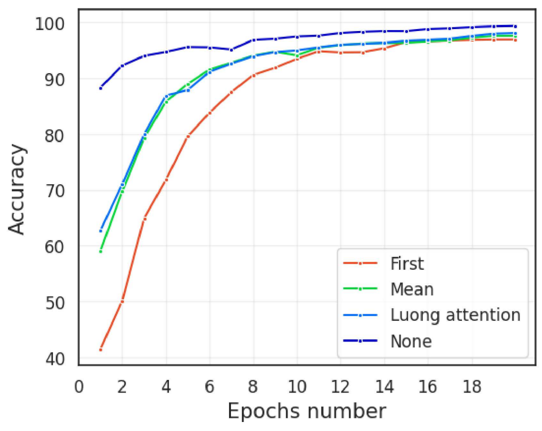

Figure 5.2: Accuracy [%] curve of the four bottlenecks analyzed.

Returning to the goal of the experiment, I decided to select as the bottleneck for the next experiments the *mean* instead of the *Luong attention* even if the second has higher accuracy, for the following reasons:

1. As table 5.3 shows the accuracy difference is very tine, around 0.5%.
2. Looking at the curves in Figures 5.1 and 5.2, the behavior in the loss and

accuracy is very similar, so both the models converge at the same speed.

3. The complexity of the model with the *mean* as the bottleneck is lower than the model with *Luong attention*.

5.2.3 Best format for input packet string

The T5 model only accepts strings as input. Therefore, the packet represented by its bytes must be converted into a string. The simplest solution is concatenating the bytes and passing the resulting string to the model. However, to help the model better understand that each pair of numbers represents a byte, it may be useful to introduce some white spaces between the bytes.
I conducted three experiments named *noSpaced*, *every4*, and *every2*. In *noSpaced*, there are no spaces between bytes, aiming to determine if ignoring white spaces aids the tokenizer. The *every4* experiment groups bytes in pairs, following the notation commonly found in the literature. Lastly, *every2* groups bytes individually, adhering to the Wireshark style.

	noSpaced	**Every4**	**Every2**
Total accuracy	89.55	**93.87**	93.69
srcIP	72.17	86.35	**87.41**
dstIP	71.81	86.54	**86.9**
IPttl	99.89	**100.0**	**100.0**
IPid	98.57	**99.79**	99.31
IPchk test	77.38	**78.41**	77.28

	noSpaced	**Every4**	**Every2**
chk3L	96.21	**99.89**	99.47
TCPwnd	99.6	**99.9**	99.8
src3L	95.61	**98.82**	98.66
TCPack	92.58	**92.68**	**92.68**
TCPseq	87.8	**89.57**	87.7
payload length	95.22	**96.47**	95.93
start payload	94.64	99.4	**99.89**

Table 5.4: Accuracy [%] for task in testing *Hybrid* dataset for three different format input strings.

Table 5.4 shows the results on the same test set for the three experiments where the training parameters are the same in terms of number of epochs, learning rate, seed, type of bottleneck, and input length. Looking at the results, it is possible to notice that the difference between the three formats is minimal. The maximum difference considering the total accuracy is 4% between *noSpaced* and *every4*, also for the subcategories only slight differences can be recognized. The highest performance is obtained with *every4* experiments but the gap with *every2* is only 0.2%.
Considering that the difference between the two types of input format is minimal in terms of performance, I decided to evaluate also the accuracy values on each subtask. Also in this case, *every4* performs slightly better than *every2* given that 9 times over 12 is the highest. Therefore, given these results, I selected the best format *every4*.

5.2.4 Assessing the impact of denoiser implementation

The rationale for introducing the denoiser is to provide the model with additional methods to learn how to represent an input packet. This experiment aims to identify the optimal training parameters to maximize the effectiveness of this pre-training task. Initially, I determined the best learning rate for the denoising task by evaluating the configuration that resulted in the greatest decrease in validation loss.

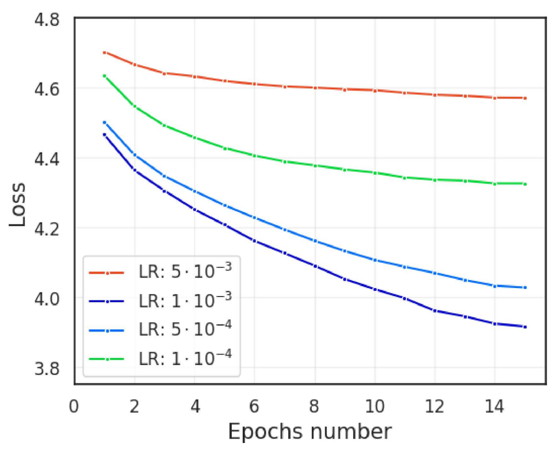

Figure 5.3: Loss curve of the denoiser validation as a function of varying learning rates, with the corruption rate set to 15%.

Figure 5.3 shows the four learning rates that are in order: $5 \cdot 10^{-3}$, $1 \cdot 10^{-3}$, $5 \cdot 10^{-4}$, and $1 \cdot 10^{-4}$. Looking at the different curves, it is immediately evident that up to $1 \cdot 10^{-3}$ there is an improvement in increasing the learning rate. Then the learning rate is too high with the consequent reduction of the sensibility of the model in learning how to denoise the input. Therefore, I selected the best $LR = 1 \cdot 10^{-3}$.
In the denoiser, a decisive parameter is the level of corruption applied to the input string. Indeed, increasing the number of tokens changed with another randomly selected, makes the task more difficult. I selected as possible corruption ratios: 0%, 15%, and 30%. The first is a simple autoencoder that must be able to reconstruct the input from its hidden representation, the second is the corruption ratio commonly adopted in the literature. The last one was introduced to see if a more challenging task can help the model learning. All configurations are compared against the baseline case without the denoiser to verify that the inclusion of Phase 0 enhances model performance.
Figures 5.4 and 5.5 show the denoiser loss and accuracy curves respectively with the different corruption rates on validation. The best value is obtained with the corruption rate equal to 15% immediately followed by the 0%. This suggests that

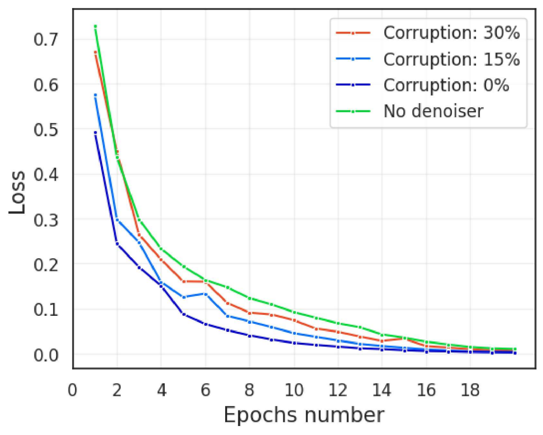

Figure 5.4: Validation loss curve for various denoiser corruption rates.

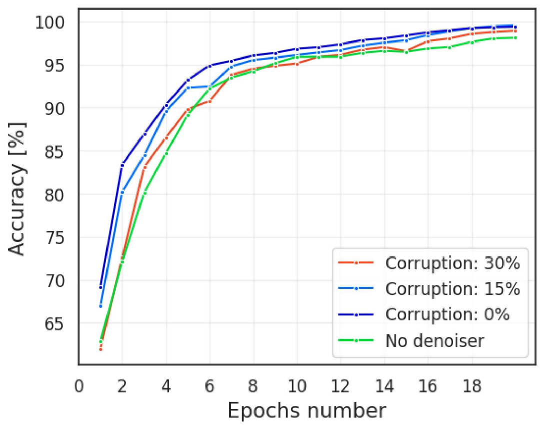

Figure 5.5: Validation accuracy curve for various denoiser corruption rates.

a small amount of noise improves the performance, but if it increases too much the behavior is reversed, such as with a rate equal to 30%. In addition, looking at the curve, I can conclude that applying the denoiser improves the learning speed since the green curve is the worst.
Table 5.5 shows the results of the test set which are slightly different. Indeed, the best model is the one that uses only the autoencoder, immediately followed by the corruption rate equal to 15% and the case without the phase 0 pre-training. This indicates that the model benefits from learning to reconstruct a packet from its internal representation, while the introduction of noise does not provide any additional advantage.

	Corruption: 30%	**Corruption: 15%**	**Corruption: 0%**	**No denoiser**
Accuracy	93.06	93.77	**94.2**	93.87

Table 5.5: Accuracy [%] on test fordifferent corruption ratio.

After all the previous considerations, my final choice is to use the denoiser and train it with a corruption rate of 0%.

5.2.5 Easy, Hybrid or Hard

The type of questions performed in the question-answering phase can deeply change how the model learns from the task. In particular, useful can be the introduction of inductive biases inside the representation that may help in identifying it.
To test the relationship among types of questions, type of traffic, and performance

I have trained three models exploiting the datasets explained in section 4.3.2. It is essential to highlight that the choice cannot be done on the question-answering task, but is performed end-to-end. This is due to the absence of the possibility of comparing three different QA tasks in questions and data. Therefore I took all the decisions at this step looking at the results of *Task2*.
First, I have explored which is the most suitable learning rate applied to the *Hybrid* dataset. In particular, I tested $1 \cdot 10^{-3}$, $5 \cdot 10^{-4}$, $1 \cdot 10^{-4}$ covering a wide range to have a macro idea of the best area of learning rates. Then, I further explored the space with smaller intervals, when the best dataset was decided. As Figure 5.6 and 5.7 show the best one is $5 \cdot 10^{-4}$. Indeed, a learning rate of $1 \cdot 10^{-3}$ is too high, causing instability in the accuracy curve due to excessive updates to the model weights, which overshoot the optimal solution. On the other hand, a learning rate of $1 \cdot 10^{-4}$ is too low, leading the model to settle into a suboptimal solution.
Even considering the test set results shown in Table 5.6, the behavior is the same, making the learning rate equal to $5 \cdot 10^{-4}$ the best choice.

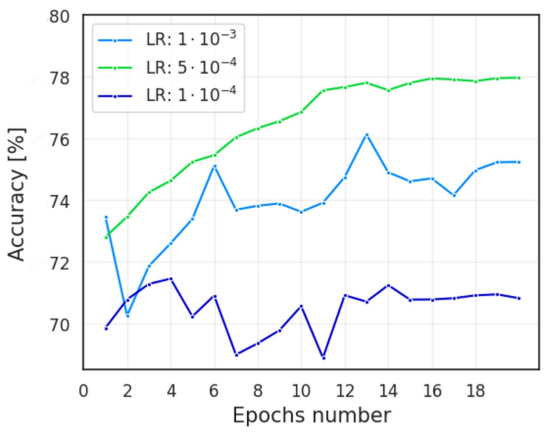

Figure 5.6: Accuracy [%] curve on validation in the *Hybrid* dataset for learning rates selection.

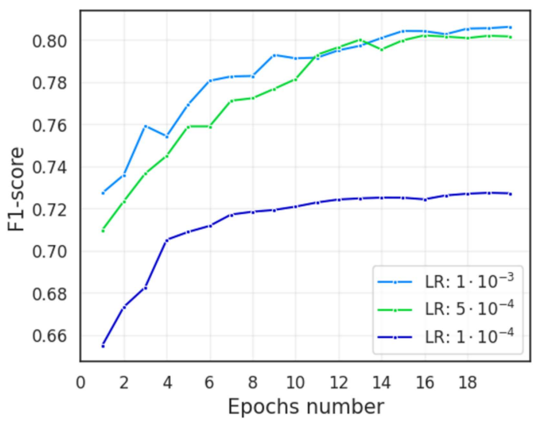

Figure 5.7: F1-score curve on validation in the *Hybrid* dataset for learning rates selection.

	LR = $1 \cdot 10^{-3}$	LR = $5 \cdot 10^{-4}$	LR = $1 \cdot 10^{-4}$
Accuracy	71.24	**74.66**	69.92
F1-score	74.84	**75.29**	71.14

Table 5.6: Accuracy [%] and F1-score on the test for different learning rates on the *Hybrid* dataset.

The model's final structure was quite decided at this point of the experiments. One of the last decisions is to select which type of questions is the best for a good packet representation for *Taks2*. Until now, the choice was the *Hybrid* dataset because selecting the most complete one was reasonable.
However, how Figure 5.8 and Figure 5.9 show the performance on the *Hybrid* dataset are outclassed by the ones with the *Easy* one. Indeed, considering the performance in the validation both for F1-score and accuracy it is higher than 5%. Concerning the *Hard* dataset, the behavior is close to the *Hybrid*, obtaining the same value for the F1-score, while slightly worse in accuracy.
Also looking at the results of the test set, shown in Table 5.10, the findings are quite the same. The *Easy* dataset achieves a sota level of accuracy and F1 score outclassing the other two. By the way, also the other two question-answering tasks are quite good. It is significant to point out that the training on *Easy* exhibits slight overfitting. The disparity in both accuracy and F1-score between the validation and test sets is notably high.

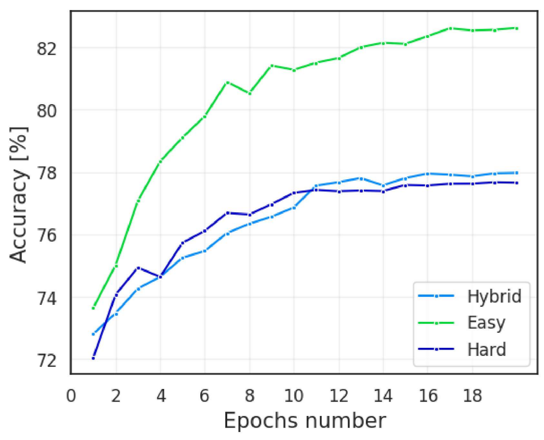

Figure 5.8: Accuracy [%] on validation for the dataset selection among *Easy*, *Hard*, and *Hybrid*.

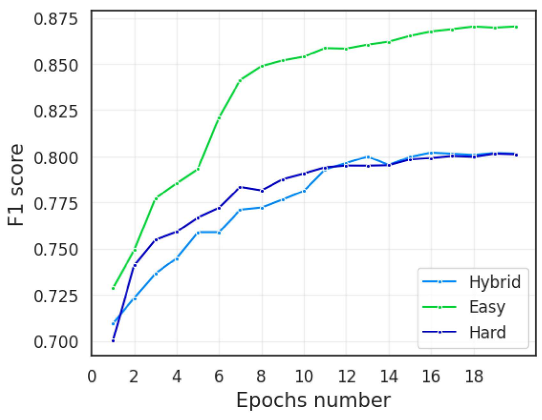

Figure 5.9: F1-score on validation for the dataset selection among *Easy*, *Hard*, and *Hybrid*.

	Easy	**Hard**	**Hybrid**
Accuracy	**77.83**	74.03	74.66
F1-score	**80.54**	74.40	75.29

Table 5.7: Accuracy [%] and F1 score on test for ifferent datasets *Easy*, *Hard*, and *Hybrid* with the parameters identified up to now.

To have a qualitative visual evaluation, I also investigated the 2D visualization of the packet representations after the classification head training with the model trained on the *Easy* dataset and the *Hard* one. To reduce the 768-dimensional representation to 2 dimensions, I applied the UMAP algorithm. This fast dimensionality-reduction technique effectively preserves the global structure of the data, particularly when reducing from high-dimensional spaces. Considering the results I expected to obtain a better *clusterization* with the *Easy* dataset than the others. However, the groups will be noisy since the score of accuracy and F1-score are not so high. As Figures 5.10 and 5.11 show, my second hypothesis is confirmed, while the first cannot be conclusively addressed through visual inspection alone. In both figures 5.10 and 5.11 the crosses represent the centroid of each cluster class and by comparing its position with respect to the other points I can make qualitative observations. First of all, we see that the *P2P* and *streaming* traffic obtain a good representation since both clusters are very concentrated close to the centroid. Although, this is not the same for the other classes, because the points are more spread.
I can conclude that the 2D visual inspection of the packet representation on the task does not provide additional information for selecting the best parameters. It only offers a qualitative assessment of the solution's effectiveness.

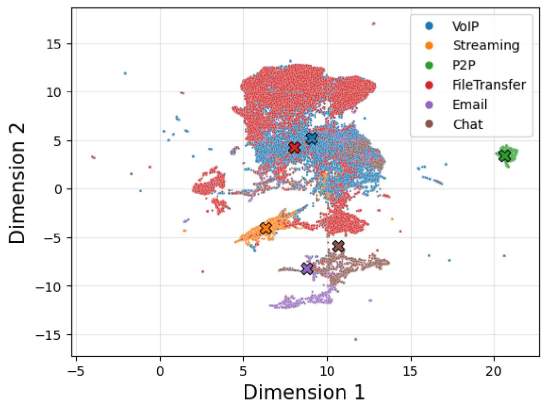

Figure 5.10: Plot in 2D of the packet representation derived from training with *Easy* question-answering dataset on *Task2* classification benchmark.

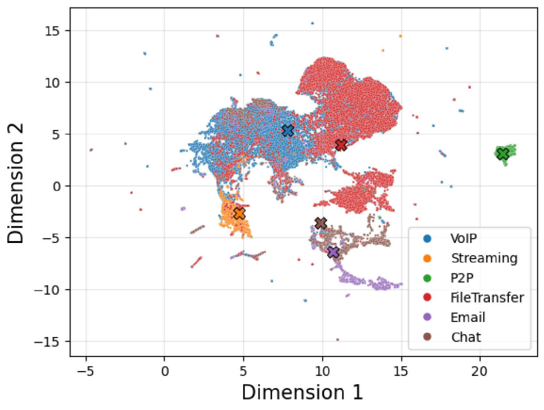

Figure 5.11: Plot in 2D of the packet representation derived from training with *Hard* question-answering dataset on *Task2* classification benchmark.

At this point, the final pipeline is outlined and the parameters of the *Task2* are fixed. Lastly, I must select the best learning rate for each classification task. Then, in the next experiments, I try to analyze some features of the model to evaluate its performance under different metrics such as training time.

5.2.6 Fine selection of the learning rate

As a final analysis, for each classification task, it is necessary to evaluate which is the best learning rate. In *Task2*, based on the observations in Section 5.2.5, I evaluated a narrower learning rate interval to determine if a slight improvement could be achieved.
So, as learning rates I tested $2.5 \cdot 10^{-4}$, $5 \cdot 10^{-4}$, $7.5 \cdot 10^{-4}$ and the results are shown in Figure 5.12 and 5.13, and in Table 5.8. Looking at the results, I noticed that the performance of the two highest learning rates is quite equal, while $2.5 \cdot 10^{-4}$ learns too slowly and always remains under the others. Comparing the two learning rates $5 \cdot 10^{-4}$ and $7.5 \cdot 10^{-4}$ on the test set is quite difficult because the first performs better in accuracy, while the latter in the F1 score. In the end, I decided to maintain the learning rate equal to $5 \cdot 10^{-4}$ because the accuracy curve in validation is more stable with respect to the other. Indeed, with the learning rate equal to $7.5 \cdot 10^{-4}$, the curve presents many segmentations, meaning that the learning rate starts to be too high.

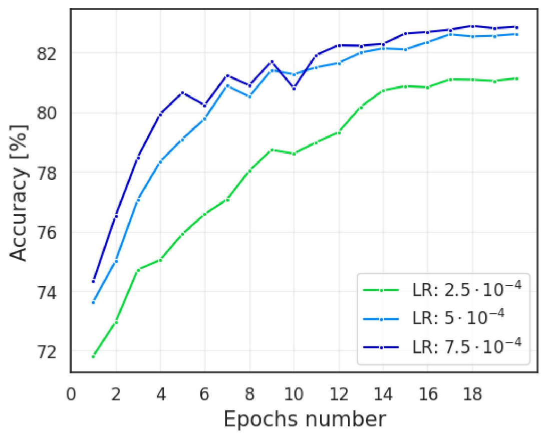

Figure 5.12: Accuracy [%] curves on validation for fine learning rate selection.

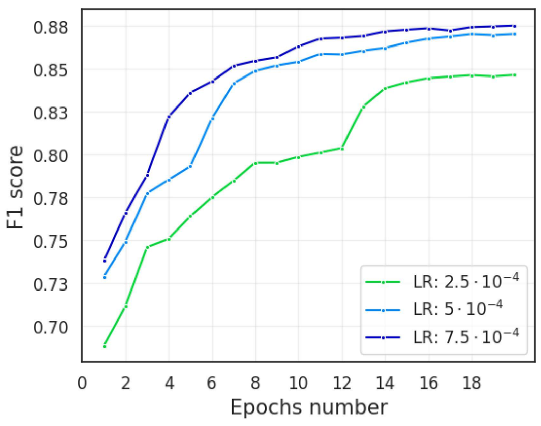

Figure 5.13: F1-score curves on validation for fine learning rate selection.

	LR $= 2.5 \cdot 10^{-4}$	LR $= 5 \cdot 10^{-4}$	LR $= 7.5 \cdot 10^{-4}$
Accuracy	74.6	**77.8**	77.7
F1-score	78.1	80.5	**80.7**

Table 5.8: Accuracy [%] and F1 score on test for different learning rates on the *Task2*.

5.2.7 Phase II: evaluation of training time

This experiment is fundamental to satisfy the objective of the entire thesis. Indeed, we require a packet representation that can be swiftly applied across a wide range of experiments, minimizing the time spent on training and the detailed dataset creation. Therefore, I need to reduce the training time as much as possible.
This experiment tries to evaluate the time needed for each part of the model to perform its work. Table 5.9 reports the average time and standard deviation of different training on the same dataset *Task2* (sec 4.2.1) and on four different learning rates ($5 \cdot 10^{-5}$, $2 \cdot 10^{-4}$, $2.5 \cdot 10^{-4}$, $5 \cdot 10^{-4}$).

	Encoder	**Classification head**
LR $= 5 \cdot 10^{-5}$	1h 13m 10s	6m 00s
LR $= 2 \cdot 10^{-4}$	1h 10m 00s	5m 20s
LR $= 2.5 \cdot 10^{-4}$	1h 17m 20s	5m 40s
LR $= 5 \cdot 10^{-4}$	1h 10m 40s	6m 00s
Descriptive statistics	**avg = 1h 13m 50s std = 2m 50s**	**avg = 5m 40s std = 20s**

Table 5.9: Training time for the encoder and classification head in one epoch considering the *Task2* dataset on one GPU of HPC cluster (Section 5.1).

Upon examining the results, I observed a significant imbalance between the training times of the encoder and the classification head, as expected. The encoder requires an average of one hour and thirteen minutes, whereas the classification head takes approximately five minutes. This discrepancy is due to the substantial difference in complexity between the two components.
However, if we aim for a fixed representation for each packet, the encoder is only necessary during the first epoch. For subsequent epochs, I can utilize the representations computed in the first epoch. This approach results in considerable time savings, with the savings increasing as the number of epochs grows.
On the other hand, if the encoder is still trained in the epochs after the first, the packet representation will change each time so it must be re-computed. In addition, in this scenario we must consider the time of backpropagation that changes the model weights: so the training time for each epoch is $\approx 3h$. The respective training times can be roughly computed as follows:

$$t_{trainFrozen} = t_{enc} + t_{classHead} \cdot n_{epochs}$$

$$t_{trainNOfrozen} = (t_{enc} + t_{encBackPropag} + t_{classHead}) \cdot n_{epochs}$$

where t_{enc}, $t_{encBackPropag}$, and $t_{classHead}$ are respectively the time for encoder, the time for encoder backpropagation, and the time for classification head training, while n_{epochs} is the number of epochs.

5.2.8 Phase II: frozen or unfrozen encoder?

Given that the classification head used is extremely simple (Section 3.4), it might be more effective to fine-tune the encoder rather than complicate the classification head. It is important to emphasize that this is as training a model completely end-to-end. It is a more extensive fine-tuning to assist the classification layer by adapting the packet representation accordingly. The big drawback that derives from section 5.2.7, is the training time that increases a lot. Indeed, as Section 5.2.7 explains, one single epoch takes around 3 hours.
As expected, Figures 5.14 and 5.15 show that during the entire training phase, the model with the encoder *unfrozen* is always better in both accuracy and F1-score respect the other. Therefore, I can conclude that without a doubt training the encoder is very useful in terms of performance. As already mentioned several times the huge drawback is the time. Indeed, considering this specific experiment where 10 epochs were only used, the encoder *unfrozen* training time is around 1 day and eight hours, while the *frozen* one is a couple of hours. In addition, I need to consider also that epoch after epoch the packet representation is not fixed, but the encoder fines the representation for the specific task.

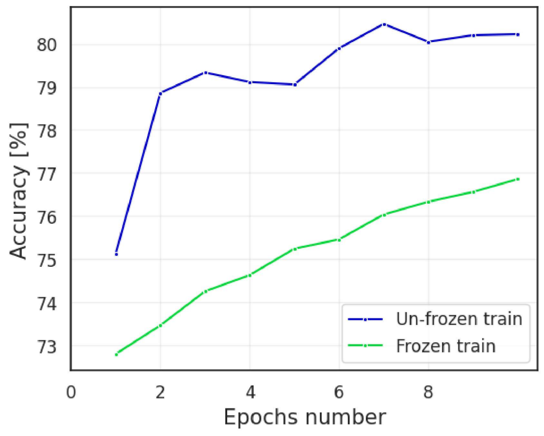

Figure 5.14: Accuracy [%] on validation with the encoder frozen/unfrozen.

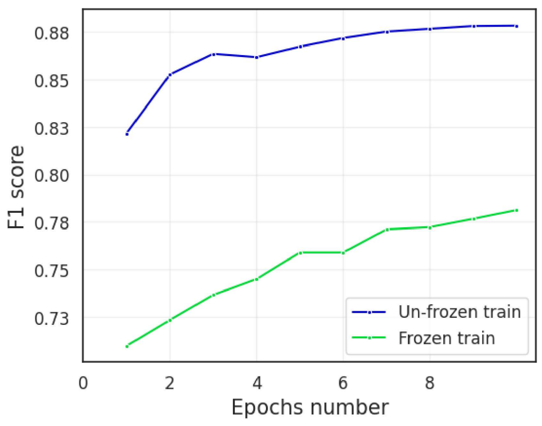

Figure 5.15: F1-score on validation with the encoder frozen/unfrozen.

Based on the previous discussions, I decided to keep the encoder *frozen* for the final results presented in the following paragraphs, while still maintaining a comparison with the *unfrozen* version.

	Frozen	**unfrozen**
Accuracy	**77.03**	76.57
F1-score	79.34	**83.60**
Training time	≈ 2h	≈ 1d 8h

Table 5.10: Accuracy [%], F1-score, and training time on test for the *frozen* and *unfrozen* versions of the model.

5.3 Results

Through the comprehensive experiments detailed in Section 5.2, I have identified the most suitable parameters. Subsequently, I need to compare the performance with state-of-the-art models and demonstrate how the packet representation encapsulates valuable information for internet traffic analysis.
As a comparison model, I selected ET-BERT [19] (Encrypted Traffic BERT), which is a sophisticated model designed for network traffic classification, particularly focusing on encrypted traffic. It exploits BERT's architecture to capture inter-datagram and inter-traffic transport relationships from large-scale, unlabeled traffic data. By pre-training on extensive datasets and fine-tuning on specific labeled encrypted traffic, ET-BERT can accurately identify various classes of network traffic in real-world scenarios.
In order to assess the impact of fine-tuning the T5 model, I also evaluated the T5 model in its original form. In this scenario, the T5 encoder is initialized with the pre-trained weights provided in the original publication.

Table 5.11 presents the results of the three models on *Task2* derived from the ISCXVPN-2016 dataset, where they classify packets into six service labels (*P2P, Email, VoIP*, etc.). For each model, both the training configuration with the encoder *frozen* or *unfrozen* are reported. These two approaches differ in the amount of learning capacity allocated to the classification model.
As Table 5.11 illustrates, our proposed model, with a carefully selected set of parameters, demonstrates a substantial improvement in performance for this specific task. When compared to the T5-base model in its default configuration, the integration of our fine-tuning process results in a marked enhancement in accuracy and nearly doubles the F1-score.
Furthermore, our model not only surpasses the T5-base model but also slightly outperforms the ET-BERT model in both accuracy and F1-score metrics. This indicates that our model's architecture and parameter optimization are highly

Model	Accuracy	F1 score	Training time
T5-base	66.7	41.7	≈ 3h
our (Frozen)	**77.8**	80.5	≈ 3h
our (unfrozen)	76.57	**83.60**	≈ 1d 8h
ET-BERT (Frozen)	74.11	75.07	≈ 2h
ET-BERT (unfrozen)	76.8	79.3	≈ 16h

Table 5.11: Comparison of the results on *Task2* in terms of accuracy, F1-score, and training-time.

effective in capturing the nuances of the task at hand.
A closer examination of the models with the encoder either *frozen* or *unfrozen* reveals a clear trend: performance metrics, including accuracy and F1-score, show noticeable improvements when the encoder is *unfrozen*. This suggests that allowing the encoder to update its weights during training enables the model to better adapt to the specific characteristics of the dataset. However, it is important to note that this performance gain comes at the cost of increased training time, highlighting a trade-off between model performance and computational efficiency.

Fundamental to a classification task is the visualization of the confusion matrix to explore the performance of the single tasks. Figures 5.16 and 5.17 show the confusion matrix for the two *frozen* models ET-BERT and ours.

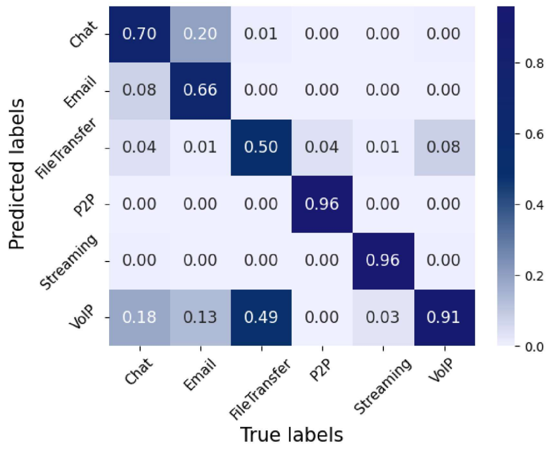

Figure 5.16: Test confusion matrix normalized on true values of OUR model on *Task2*.

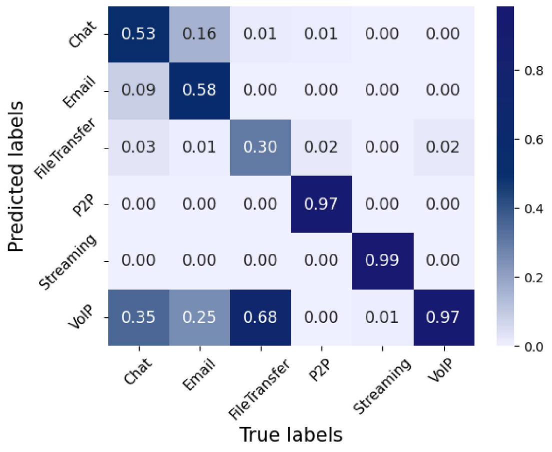

Figure 5.17: Test confusion matrix normalized on true values of ET-BERT model on *Task2*.

Upon comparing the two normalized confusion matrices, it becomes evident that the classes *P2P*, *Streaming*, and *VoIP* are the most easily recognizable by both models. This indicates a higher accuracy in identifying these types of traffic. However, there are notable areas of confusion, particularly with the *FileTransfer* class, frequently misclassified as *VoIP*. This misclassification is especially pronounced in the ET-BERT model.
Additionally, both our model and the ET-BERT model exhibit significant confusion between the *Chat* and *Email* classes. This suggests that these two types of traffic share similar characteristics that make them difficult to distinguish.
A broader analysis of the confusion matrices reveals that a substantial portion of the traffic is incorrectly identified as *VoIP*. Specifically, more than half of the *FileTransfer* traffic in our model and approximately two-thirds in the ET-BERT model are misclassified as *VoIP*. This pattern can be attributed to the class distribution detailed in Section 5.2, where *VoIP* is the most prevalent class. This may cause a bias in the model, that if in doubt, predicts *VoIP*.

One of the most relevant contribution of this thesis lies in the numerical representation of internet packets, which can be efficiently obtained after fine-tuning the model. Although powerful models are frequently proposed in the literature, deriving a meaningful representation from them is not always straightforward.

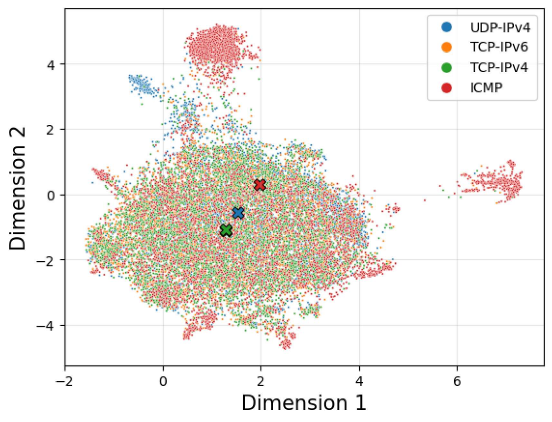

Figure 5.18: Plot in 2D of the packet representation derived from training with *Easy* question-answering dataset.

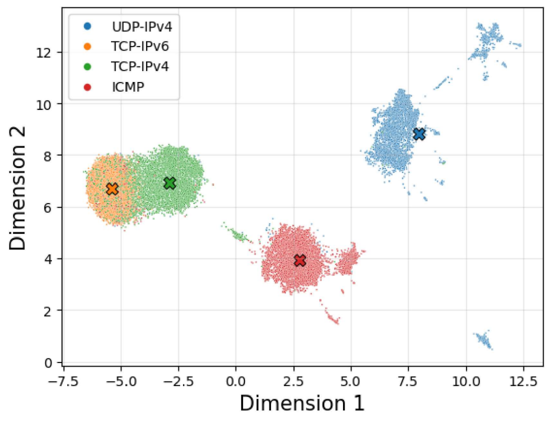

Figure 5.19: Plot in 2D of the packet representation derived from training with *Hard* question-answering dataset.

Figures 5.18 and 5.19 illustrate how valuable information for downstream tasks can be embedded within the packet representation without requiring supervised learning. For instance, by training the model with the challenging question-answering task

(*Hard* dataset), we expected that the resulting representation would encapsulate information related to the packet protocol. This expectation is confirmed by the data shown in Figure 5.19.
On the other hand, this does not suggest that the numerical representations depicted in Figure 5.18 are devoid of valuable information. Rather, they do not capture the specific feature related to the packet protocol but may instead encapsulate other aspects that are more beneficial for the downstream task.
What Figure 5.19 shows is very valuable in internet traffic analysis. Indeed, the plot demonstrates that the model can distinguish the protocols and also gives an idea of how much they differ. Indeed, I recognize three macro areas that are one of TCP, one of UDP, and the last more fragmented ICMP. Further, we can clearly recognize the two poles that distinguish IPv4 from IPv6 in the TCP area by looking at the centroids.

Chapter 6

Conclusion and future work

6.1 Overview

Throughout the thesis, I have endeavoured to address the research question: can the text analysis capabilities of large language models (LLM), replace traditional tools allowing larger-scale and automatic internet packet analysis? To explore this question, I proposed a comprehensive training pipeline based on the T5-base model, designed to generate valuable packet representations.
The fine-tuning process applied to the model was twofold. Initially, it involved an autoencoder architecture, where the objective was to reconstruct an input packet through a compressed internal representation. Subsequently, the model was fine-tuned using a question-answering approach, further enhancing its ability to understand and process packet data.
To evaluate the effectiveness of the generated representations, the model was employed to solve a classification task using the ISCXVPN-2016 dataset.
The results observed from the experiments indicate that the proposed approach is successful. The comparative analysis of the experimental findings revealed that the values for both accuracy and F1-score are on par with state-of-the-art models. Moreover, in alignment with the research question aimed at automating network packet analysis, our tool has shown the ability to produce representations that inherently embed information related to the original packets, such as protocols, without requiring explicit supervision.

Overall, the work presented in this thesis highlights the potential of large language models (LLMs) to revolutionize the field of network packet analysis by offering scalable and automated solutions. Although the initial steps have been successful, a considerable amount of work remains to be done before we can definitively answer the research question positively or negatively.

6.2 Limitations and future approach

Considering that the thesis is the first exploration of a field not widely considered, the work to get a global idea of the manner is still long. I followed a possible logical decision flow, but a more detailed analysis was possible at each step.
Firstly, in the near future, it is essential to expand the validation of the classification benchmark to encompass additional tasks. This will allow us to assess the robustness of the training pipeline across various classification scenarios.
Then there are many possible additional studies and improvements. A complete analysis of how the datasets used affect the performance can be fundamental. I already tried with section 4.3.2 to see some differences, but the used data are not so recent, the dataset size is limited, only a part of the traffic is real, and the considered protocols are only a small subset of the actual ones. Another crucial aspect concerning internet packets is evaluating how to input them into the model, such as determining whether the payload enhances the model's performance.
Another core point is the bottleneck choice that can be further extended to more complex and ad-hoc solutions that exploit the attention mechanisms. In this way, a better packet representation can be obtained. On the same idea, the classification head can be improved but maintain a similar training time. In the thesis, I implemented an elementary one, but many other efficient alternatives can be found in the literature [50].
Last but not least, the amount of fields in which the packet representation was applied. In the thesis, I evaluated the performance on a restricted number of classification tasks. In the future, it is possible to expand the classification benchmark easily, but also use the representation to solve other tasks, such as novelty and intrusion detection, or synthetic packet generation.
While this project primarily focuses on saving training time, it also opens up future possibilities for optimizing storage space for logs. These potentials, along with numerous other advancements, highlight the scope for further improvements and future research.

Bibliography

[1] Ericsson. *Ericsson Mobility Report Q2 2024 Update*. 2024. URL: `https://www.ericsson.com/4a4b71/assets/local/reports-papers/mobility-report/documents/2024/ericsson-mobility-report-q2-2024-update.pdf` (cit. on p. 1).

[2] Mahmoud Abbasi, Amin Shahraki, and Amir Taherkordi. «Deep Learning for Network Traffic Monitoring and Analysis (NTMA): A Survey». In: *Computer Communications* 170 (2021), pp. 19–41 (cit. on pp. 1, 5).

[3] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. «New Directions in Automated Traffic Analysis». In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 3366–3383. URL: `https://doi.org/10.1145/3460120.3484758` (cit. on p. 2).

[4] Xuying Meng, Yequan Wang, Runxin Ma, Haitong Luo, Xiang Li, and Yujun Zhang. «Packet Representation Learning for Traffic Classification». In: *The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2022, pp. 3546–3554 (cit. on pp. 2, 19).

[5] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Mamun, and Ali A. Ghorbani. «Characterization of Encrypted and VPN Traffic Using Time-Related Features». In: *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*. Rome, Italy: SciTePress, 2016, pp. 407–414 (cit. on pp. 3, 28).

[6] Peyman Kabiri and Ali Ghorbani. «Research on Intrusion Detection and Response: A Survey». In: *International Journal of Network Security* 1 (Jan. 2005), pp. 84–102 (cit. on p. 4).

[7] Tao Wang and Ian Goldberg. «Improved website fingerprinting on Tor». In: *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. ACM. 2013, pp. 201–212 (cit. on p. 4).

[8] Vijayanand Thangavelu, Dinil Mon Divakaran, Rishi Sairam, Suman Sankar Bhunia, and Mohan Gurusamy. «DEFT: A Distributed IoT Fingerprinting Technique». In: *IEEE Internet of Things Journal* 6.1 (2018), pp. 940–952 (cit. on p. 4).

[9] Satish Kumar, Sunanda Gupta, and Sakshi Arora. «Research Trends in Network-Based Intrusion Detection Systems: A Review». In: *IEEE Access* 9 (2021), pp. 157761–157779 (cit. on p. 4).

[10] Mahdi Zamani and Mahnush Movahedi. *Machine Learning Techniques for Intrusion Detection.* 2015. arXiv: `1312.2177 [cs.CR]`. URL: `https://arxiv.org/abs/1312.2177` (cit. on p. 4).

[11] Andrew W. Moore and Denis Zuev. «Internet traffic classification using bayesian analysis techniques». In: *SIGMETRICS Perform. Eval. Rev.* 33.1 (June 2005), pp. 50–60. ISSN: 0163-5999. URL: `https://doi.org/10.1145/1071690.1064220` (cit. on p. 5).

[12] Tom Auld, Andrew W. Moore, and Stephen F. Gull. «Bayesian Neural Networks for Internet Traffic Classification». In: *IEEE Transactions on Neural Networks* 18.1 (2007), pp. 223–239 (cit. on p. 5).

[13] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. «Website Fingerprinting at Internet Scale». In: *Network and Distributed System Security Symposium.* 2016. URL: `https://api.semanticscholar.org/CorpusID:15302617` (cit. on p. 5).

[14] Jiong Zhang, Mohammad Zulkernine, and Anwar Haque. «Random-Forests-Based Network Intrusion Detection Systems». In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.5 (2008), pp. 649–659 (cit. on p. 5).

[15] Raouf Boutaba, Mohammad A. Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M. Caicedo. «A comprehensive survey on machine learning for networking: evolution, applications and research opportunities». In: *Journal of Internet Services and Applications* 9.1 (2018), p. 16. DOI: `10.1186/s13174-018-0087-2` (cit. on p. 5).

[16] Sonni-Ali Miller, Jason A. White, Rupak Chowdhury, Dominique N. Gales, Berhanu Tameru, Amit K. Tiwari, and Temesgen Samuel. «Effects of consumption of whole grape powder on basal NF-B signaling and inflammatory cytokine secretion in a mouse model of inflammation». In: *Journal of Nutrition & Intermediary Metabolism* 11 (2018), pp. 1–8. ISSN: 2352-3859. URL: `https://www.sciencedirect.com/science/article/pii/S2352385917302542` (cit. on p. 5).

[17] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. «Malware traffic classification using convolutional neural network for representation learning». In: *2017 International Conference on Information Networking (ICOIN)*. IEEE. 2017, pp. 712–717 (cit. on p. 5).

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: `1706.03762 [cs.CL]`. URL: `https://arxiv.org/abs/1706.03762` (cit. on pp. 5, 8).

[19] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. «ET-BERT: A Contextualized Datagram Representation with Pre-training Transformers for Encrypted Traffic Classification». In: *Proceedings of the ACM Web Conference 2022*. WWW '22. ACM, Apr. 2022 (cit. on pp. 5, 48).

[20] Ruijie Zhao, Mingwei Zhan, Xianwen Deng, Yanhao Wang, Yijun Wang, Guan Gui, and Zhi Xue. «Yet Another Traffic Classifier: A Masked Autoencoder Based Traffic Transformer with Multi-Level Flow Representation». In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.4 (June 2023), pp. 5420–5427. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/25674` (cit. on p. 5).

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023 (cit. on pp. 5, 11–13, 19).

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: `1810.04805 [cs.CL]`. URL: `https://arxiv.org/abs/1810.04805` (cit. on pp. 6, 13).

[23] Tom B Brown et al. *Language Models are Few-Shot Learners*. 2020. URL: `https://arxiv.org/abs/2005.14165` (cit. on p. 6).

[24] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: `2303.08774 [cs.CL]`. URL: `https://arxiv.org/abs/2303.08774` (cit. on p. 6).

[25] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: `2302.13971 [cs.CL]`. URL: `https://arxiv.org/abs/2302.13971` (cit. on p. 6).

[26] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. «Extracting and composing robust features with denoising autoencoders». In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 1096–1103. ISBN: 9781605582054. URL: `https://doi.org/10.1145/1390156.1390294` (cit. on p. 10).

[27] Ivan Habernal, Omnia Zayed, and Iryna Gurevych. «C4Corpus: Multilingual Web-size Corpus with Free License». In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16).* European Language Resources Association (ELRA), May 2016, pp. 914–922 (cit. on p. 13).

[28] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. *SQuAD: 100,000+ Questions for Machine Comprehension of Text.* 2016. arXiv: `1606.05250 [cs.CL]`. URL: `https://arxiv.org/abs/1606.05250` (cit. on p. 13).

[29] U. Fuchs. «A Short Introduction to Computer Networks». In: *Molecular Imaging: Computer Reconstruction and Practice.* Ed. by Y. Lemoigne and A. Caner. NATO Science for Peace and Security Series B: Physics and Biophysics. Springer, Dordrecht, 2008. DOI: `10.1007/978-1-4020-8752-3_10` (cit. on p. 14).

[30] Muzhir Al-Ani and Rola A.A.Haddad. «IPv4/IPv6 Transition». In: *International Journal of Engineering Science and Technology* 4 (Dec. 2012), pp. 4815–4822 (cit. on p. 15).

[31] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation.* 2015. arXiv: `1508.04025 [cs.CL]`. URL: `https://arxiv.org/abs/1508.04025` (cit. on p. 20).

[32] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. «A Comprehensive Overview of Large Language Models». In: (2024). arXiv: `2307.06435 [cs.CL]`. URL: `https://arxiv.org/abs/2307.06435` (cit. on p. 24).

[33] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. «MAWILab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking». In: *ACM CoNEXT '10.* Philadelphia, PA, Dec. 2010 (cit. on p. 24).

[34] Malware Traffic Analysis. *About Malware Traffic Analysis.* Accessed: 2024-08-27. 2024. URL: `https://malware-traffic-analysis.net/about.html` (cit. on pp. 24, 27).

[35] Internet Initiative Japan Inc. «IIJ Vol.173». In: (2024). Accessed: 2024-08-21. URL: `https://www.iij.ad.jp/news/iijnews/vol_173/detail_08.html` (cit. on p. 25).

[36] Romain Fontugne, Patrice Abry, Kensuke Fukuda, Darryl Veitch, Kenjiro Cho, Pierre Borgnat, and Herwig Wendt. «Scaling in Internet Traffic: a 14 year and 3 day longitudinal study, with multiscale analyses and random projections». In: (2017). arXiv: `1703.02005 [cs.NI]`. URL: `https://arxiv.org/abs/1703.02005` (cit. on p. 25).

[37] MAWI Working Group. *MAWI Working Group Traffic Archive Guidelines.* Accessed: 2024-08-27. 2024. URL: `https://mawi.wide.ad.jp/mawi/guideline.txt` (cit. on p. 25).

[38] Nour Moustafa and Jill Slay. «UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)». In: Nov. 2015. DOI: `10.1109/MilCIS.2015.7348942` (cit. on p. 26).

[39] G. Memik, W.H. Mangione-Smith, and W. Hu. «NetBench: a benchmarking suite for network processors». In: *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281).* 2001, pp. 39–42 (cit. on p. 28).

[40] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. «Characterization of Tor Traffic Using Time Based Features». In: *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP).* Porto, Portugal: SciTePress, 2017, pp. 253–262 (cit. on p. 28).

[41] Wei Wang, Ming Zhu, Xuewen Zeng, Xiaozhou Ye, and Yiqiang Sheng. «Malware Traffic Classification Using Convolutional Neural Network for Representation Learning». In: *Proceedings of the International Conference on Information Networking (ICOIN).* IEEE, 2017, pp. 712–717 (cit. on p. 28).

[42] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J. Dubois, Martina Lindorfer, David Choffnes, Maarten van Steen, and Andreas Peter. «FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic». In: *Proceedings of the Network and Distributed System Security Symposium (NDSS).* Vol. 27. Internet Society, 2020 (cit. on p. 28).

[43] Mohammadreza MontazeriShatoori, Logan Davidson, Gurdip Kaur, and Arash Habibi Lashkari. «Detection of DoH Tunnels Using Time-Series Classification of Encrypted Traffic». In: *Proceedings of the 2020 International Conference on Cyber Science and Technology Congress (CyberSciTech).* IEEE, 2020, pp. 63–70 (cit. on p. 28).

[44] Euclides Carlos Pinto Neto, Sajjad Dadkhah, Raphael Ferreira, Alireza Zohourian, Rongxing Lu, and Ali A. Ghorbani. «CICIoT2023: A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment». In: *Sensors* 23.13 (2023), p. 5941 (cit. on p. 29).

[45] Philippe Biondi. *Scapy.* Version 2.4.5. 2024. URL: `https://scapy.net` (cit. on pp. 30, 31).

[46] Matteo Boffa, Giulia Milan, Luca Vassio, Idilio Drago, Marco Mellia, and Zied Ben Houidi. «Towards NLP-based Processing of Honeypot Logs». In: *2022 IEEE European Symposium on Security and Privacy Workshops (EuroSPW).* 2022, pp. 314–321. DOI: `10.1109/EuroSPW55150.2022.00038` (cit. on p. 35).

[47] Matteo Boffa, Idilio Drago, Marco Mellia, Luca Vassio, Danilo Giordano, Rodolfo Valentim, and Zied Ben Houidi. «LogPrécis: Unleashing language models for automated malicious log analysis: Précis: A concise summary of essential points, statements, or facts». In: *Computers Security* 141 (2024), p. 103805. URL: `https://www.sciencedirect.com/science/article/pii/S0167404824001068` (cit. on p. 35).

[48] *Legion Cluster - HPC@PoliTO.* Accessed: 2024-09-29. URL: `https://www.hpc.polito.it/legion_cluster.php` (cit. on p. 35).

[49] *Computing Facilities - SmartData@PoliTO.* Accessed: 2024-09-29. URL: `https://smartdata.polito.it/computing-facilities/` (cit. on p. 35).

[50] Zhuoyi Yang, Ming Ding, Yanhui Guo, Qingsong Lv, and Jie Tang. *Parameter-Efficient Tuning Makes a Good Classification Head.* 2023. arXiv: `2210.16771 [cs.CL]`. URL: `https://arxiv.org/abs/2210.16771` (cit. on p. 53).