UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Classification of Honeypot Data Using the MITRE Framework

Author:
Hug Camps i Regàs

Supervisor:
Dr. Eloi Puertas i Prats

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

June 30, 2025

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Classification of Honeypot Data Using the MITRE Framework

by Hug Camps i Regàs

Proactive cybersecurity measures are essential for effective risk mitigation in increasingly complex and evolving digital environments. Achieving this requires not only the collection of relevant data but also its accurate interpretation and the development of specialized analytical frameworks. This project focuses on addressing the challenge of interpreting cyber threat data by classifying honeypot data, provided by the Global Cyber Alliance (GCA), according to the MITRE ATT&CK Matrix—a widely recognized framework for understanding adversarial behavior. In an era dominated by large language models (LLMs), we investigate an alternative approach based on smaller, specialized models. Specifically, we design a custom architecture of lightweight models and train them for the task, evaluating their performance across various configurations. Our findings demonstrate that these models can, in certain scenarios, outperform larger LLMs in both accuracy and efficiency, offering a more sustainable and cost-effective solution for targeted cybersecurity applications.

This thesis used ChatGPT to improve language clarity and style of author's preliminary drafts. All data analysis, interpretation, and argumentation were conducted by the author.

Acknowledgements

I would like to express my gratitude to my thesis tutor, Eloi, whose guidance and support were especially valuable when things became challenging.

I want to thank to the Agència de Ciberseguretat de Catalunya for giving me the opportunity to step into the world of cybersecurity. In particular, I am grateful to Ana and Santi for their trust.

Finally, I want to thank my flatmates for the support given even in the most stressful moments.

Contents

Chapter 1

Introduction

Cybersecurity continues to be a critical domain of concern as the complexity and scale of digital infrastructures increase globally. With the growing interconnectivity of systems and the proliferation of internet-facing services, the number of potential attack vectors has expanded significantly. In response, organizations and governments have intensified their focus on proactive cyber defense mechanisms, particularly those aimed at early detection and rapid mitigation of threats.

In response to this threats in the cybersecurity landscape, the Global Cybersecurity Alliance (GCA)—a non-profit organization dedicated to enhancing online safety—has developed the Automated IoT Defence Ecosystem (AIDE) project. Recognizing the limitations of reactive security measures and the growing need for proactive threat intelligence sharing, AIDE aims to establish an automated platform for the collection, analysis, and dissemination of actionable data on attacks targetingcomputers. This innovative approach seeks to foster a more resilient and secure ecosystem through collaborative threat intelligence and the potential for automated defense responses.

This project, carried out at the Agència de Ciberseguretat de Catalunya (ACC), is developed within the AIDE framework. Almost all data utilized originates from GCA's collection efforts, and our focus lies in analyzing and leveraging this data to develop practical tools that support the day-to-day mitigation work performed by ACC. This is especially valuable given that ACC is a public agency, meaning improvements here can have a direct impact on other critical sectors, such as healthcare and education, which are frequent targets of cyberattacks.

Building upon those findings, the current iteration of the project narrows its focus to the detailed analysis of command-line activity observed in honeypot environments. This shift is driven by the recognition that command sequences executed by adversaries often reveal valuable information about their tactics, techniques, and procedures (TTPs). By accurately classifying such commands, defenders can enhance both their preventive and responsive capabilities.

This thesis presents the design and implementation of a system aimed at classifying command-line activity within the framework of MITRE ATT&CK—a widely adopted taxonomy of adversarial behaviors. The work emphasizes not only the technical challenges of classification but also the practical aspects of building a scalable and cost-effective solution. In particular, the project explores the limitations of large language models (LLMs) in this context and motivates the development of a specialized classification pipeline tailored to the task.

The next section outlines the specific objectives of the project, which align with the strategic goals of the ACC and serve as the foundation for the work presented in the remainder of this thesis.

1.1 Objectives

In last year's edition of the AIDE project, the ACC presented a paper based on data collected from honeypots (Rodríguez et al., 2024). That work provided a detailed analysis of the data, focusing on characteristics such as the geographical origin of attacks and the types of passwords used by adversaries. It then applied machine learning techniques to perform clustering, followed by a time-series analysis of the attack patterns.

Building upon that foundation, this year's project continues in the direction of analyzing and classifying command-line activity, with a specific focus on supporting attack mitigation. To this end, the current project has two main objectives:

- To conduct an extensive analysis of honeypot command data, with a particular focus on mapping it to MITRE ATT&CK tactics.
- To develop an interactive interface—such as a chatbot—that, when given a list of commands representing a potential attack, can provide insightful classification and analysis.

These two objectives are designed to enhance two core functions of the ACC:

- Action and Prevention
- Incident Response

To achieve both objectives, we need to develop a model capable of classifying individual commands within the MITRE ATT&CK framework, and this will be the part described in this Final Master's Thesis. Our initial idea was to leverage an OpenAI model (e.g., GPT-4o, OpenAI, 2024), but early tests revealed suboptimal performance. Similar results were observed with other large language models.

Our hypothesis is that the poor performance stems from several key limitations:

- These models are primarily optimized for natural language generation tasks. While they have demonstrated improving capabilities in understanding code, they still struggle with accurately interpreting command-line syntax and semantics.
- The classification challenge itself is highly complex: the MITRE ATT&CK framework defines over 14 distinct tactics. This creates an inherently imbalanced classification problem, as it is difficult to provide equal training data for each class. As a result, the models tend to produce biased predictions.

Another significant limitation of using commercial APIs like OpenAI is cost. Since the final goal of the project is to classify potentially millions of commands for in-depth prevention analysis, relying on a paid API becomes economically unsustainable.

Given these constraints, we decided not to focus on improving the performance of existing large language models using techniques such as in-context learning or chain-of-thought prompting. Instead, we opted to develop our own tailored model. By designing a custom pipeline suited to our specific dataset and classification needs, we aim to build a more reliable and cost-efficient solution.

At the end of the project, we plan to compare the performance of our custom model against commercial LLMs to validate our hypothesis and assess whether the specialized model offers a tangible improvement in both accuracy and practicality.

Chapter 2

Preliminaries

2.1 MITRE ATT&CK Framework

As previously mentioned, cybersecurity is becoming increasingly complex and deeply embedded in our daily lives. Addressing this challenge requires not only data to identify patterns but also new methods and structured frameworks to interpret them. The MITRE ATT&CK Framework, developed by the non-profit MITRE, fulfills this need. It is a knowledge base of adversarial tactics and techniques, grounded in real-world observations. ATT&CK emphasizes how adversaries interact with systems during an attack, covering various stages of the adversary lifecycle and the platforms they target.

For this project, we specifically utilize the MITRE ATT&CK Matrix—a structured representation of known adversarial behavior in enterprise IT environments. This matrix is a core component of the broader ATT&CK Framework and focuses on the tactics and techniques used by threat actors to infiltrate and operate within systems such as Windows, Linux, macOS, and cloud platforms (e.g., AWS, Azure, Google Cloud).

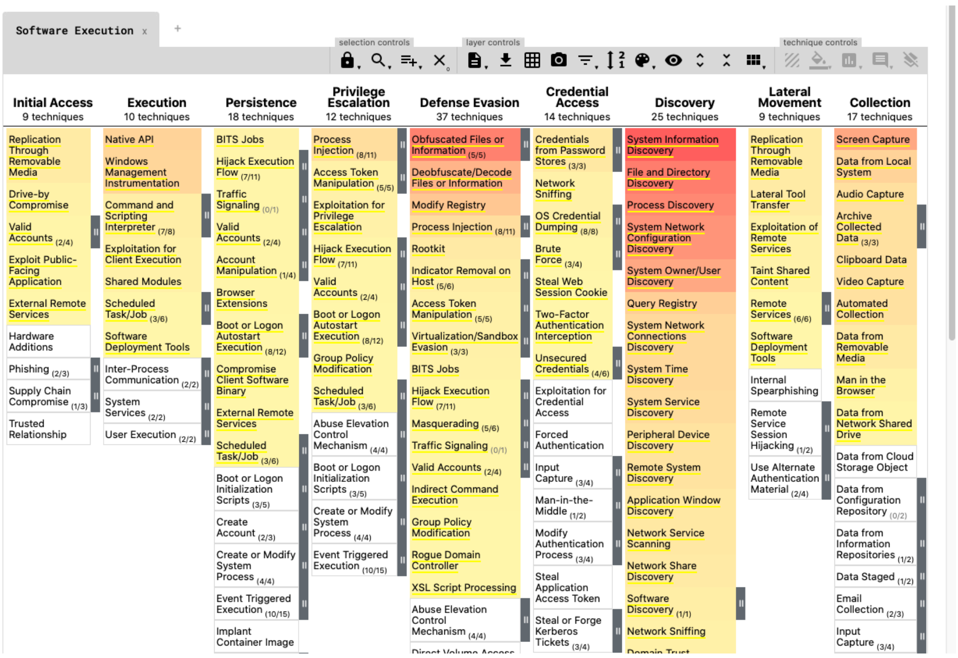


FIGURE 2.1: MITRE Matrix representation, where every column is a tactic with all its techniques below.

To understand better this system we have to think of it not as a matrix with rows and columns representing features but as a dictionary of dictionaries (visualization in Figure 2.1). It is organized around 14 high-level tactics: Reconnaissance, Resource Development, Initial Access, Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection, Command and Control, Exfiltration, and Impact. These tactics represent the adversary's objectives. For example, the tactic **Initial Access** is described as follows:

> **Initial Access**
> The adversary is trying to get into your network.
> Initial Access consists of techniques that use various entry vectors to gain a foothold within a network. Techniques include spearfishing and exploiting vulnerabilities on public-facing web servers. Gained access may be used for continued infiltration through valid credentials or external remote services, or may be short-lived due to changing access controls.

Within each tactic are specific techniques. For **Initial Access**, there are 11 techniques, including: Content Injection, Drive-by Compromise, Exploit Public-Facing Application, External Remote Services, Hardware Additions, Phishing, Replication Through Removable Media, Supply Chain Compromise, Trusted Relationship, Valid Accounts, and Wi-Fi Compromise.

Here is an example of one of these techniques:

> **Content Injection**
> Adversaries may gain access and maintain communication with victims by injecting malicious content into systems via online network traffic. Instead of tricking users into visiting malicious sites, adversaries may exploit compromised communication channels to manipulate or inject content directly.
>
> Methods of content injection include:
>
> - **From the middle:** where the adversary intercepts and alters legitimate client-server communications (distinct from Adversary-in-the-Middle techniques focused on enterprise environments).
>
> - **From the side:** where malicious content races the legitimate server's response to reach the client first.
>
> Such attacks often exploit upstream infrastructure, such as an Internet Service Provider (ISP), as seen in lawful interception scenarios.

This example illustrates how techniques describe the means used to achieve each tactic's objective. Because techniques delve into the actual methodologies employed by adversaries, they could form the foundation of our classification system for attacks, or even sub-techniques, which offer even more granular detail. However, we will omit them at this stage to avoid unnecessary complexity, and will focus in the 14 tactics.

The MITRE documentation primarily provides natural language descriptions and examples of tools or methods used for each technique. One of the project's key design challenges is the absence of lower-level indicators (such as specific command-line inputs), which complicates direct mapping from data to techniques.

Nonetheless, MITRE ATT&CK is an internationally recognized, data-driven framework that greatly facilitates the task of attack analysis and mitigation. This makes it a valuable resource for the broader cybersecurity community, particularly for those working with honeypot data as in the AIDE project. It also enables us to deliver more impactful tools for public institutions like the ACC.

2.2 Understanding the Command Line

The **command line interface (CLI)** is a text-based interface used to interact with operating systems and software by typing commands. CLIs require users to input specific instructions using a keyboard. This form of interaction offers a high degree of control and flexibility, particularly valued by system administrators, developers, and attackers alike.

A typical command-line input consists of:

- A **command**: the main action to be performed (e.g., `ls`, `cd`, `curl`).
- One or more **options/flags**: parameters that modify the behavior of the command (e.g., `-l`, `-verbose`).
- **Arguments**: additional information or targets for the command (e.g., file paths, URLs).

For example, the command:

```
/bin/busybox wget http://malicious.example.com/payload.sh
```

uses `busybox` to invoke the `wget` utility and download a script from a remote server. `BusyBox` is a single executable that provides minimalist versions of common UNIX utilities, often used in embedded systems. In many attacks, adversaries exploit BusyBox to run commands on compromised systems where traditional binaries may not be available.

Understanding and interpreting this command-line activity is therefore essential for accurate threat classification and effective response.

Chapter 3

Data

3.1 Collection of the Data: Honeypots and Honeyfarms

Honeypots are network-connected systems designed to collect information about the intent, methods, and origins of cyberattacks. These systems are purposefully crafted to closely resemble real environments, presenting themselves as legitimate targets to adversaries and encouraging them to reveal their full range of tools and techniques.

To achieve this, the GCA operates a honeyfarm consisting of more than 197 honeypots distributed across 25 countries. These use ProxyPot honeypots to collect real-time attack traffic. The standout feature of this technology is its ability to proxy incoming traffic to actual devices, allowing the system to record full, unencrypted Packet Capture files (PCAPs) for in-depth analysis. ProxyPot performs high-level, automated analysis of these PCAPs, extracting valuable information across a wide range of features. It supports multiple protocols, including FTP, HTTP, HTTPS, ICMP, SFTP, SSH, and Telnet.

The data we receive is highly diverse. As shown in Table 3.1, one of the key fields is `allCommands`, which lists all commands executed during a session. We will limit our analysis to sessions that include at least one executed command.

TABLE 3.1: Some relevant fields of AIDE Data Using ProxyPot Technology

Field	**Description**	**Field**	**Description**
_id	Unique document identifier	_index	Document index name
@timestamp	Event recording time (ISO8601 UTC)	session	Unique session identifier
sessionLength	Session duration (`endTime - startTime`)	startTime	Session start timestamp
endTime	Session end timestamp	protocol	Communication protocol used
clientVersion	Client software version	clientIP	Intruder IP address
clientPort	Intruder source port	as_org	Organization associated with IP
asn	Autonomous System Number	city_name	City location

country_name	Country name	category	Session classification (Scan/Attempt/Intrusion)
allCommands	All executed commands	commands	Fully emulated Unix commands
hashes	Downloaded malware file hashes	virustotal	VirusTotal analysis results
urls	Accessed URLs	country_code2	Two-letter country code
country_code3	Three-letter country code	dma_code	Designated Market Area code
ip	IP address	latitude	Location latitude
longitude	Location longitude	timezone	IP timezone

The deployment of ProxyPot technology in the honeyfarm began in November 2024. As mentioned, this upgrade enhances the realism of the emulated systems, leading to improved coverage of real-world attack traffic. Earlier data, which has already been studied in works such as Cristian Munteanu, 2021 and Rodríguez et al., 2024, would lead to different conclusions and is excluded for consistency.

It is important to note that, despite these improvements, sophisticated adversaries may still identify honeypots after a few steps. Therefore, we expect the data to reflect real-world attack patterns within those limitations.

3.2 First sight

To perform this analysis, we use attack data collected from November 2024 to January 2025—covering a full three months. During this period, considering only attacks that include at least one executed command, we have a total of 8195845 entries.

With a first view of the data we can see a lot of commands are repeated or almost the same. In fact, if we count the number of different commands that we have this number reduces to 72365, with some commands appearing a lot of times (the most common one appears 2370649 times, for example).

To avoid having to classify those huge amount of data, making it pass thorough two different models we will normalize it by deleting non important information. We used a function designed to preprocess and normalize command-line strings found in our raw data. This standardization supports cleaner analysis and machine learning by ensuring consistent formatting and reducing noise. Almost all functions we used to build it were from Rodríguez et al., 2024 work.

This function performs the following transformations:

1. **Bracket Removal:** Square brackets `[ ]` are stripped using regex to clean enclosing structures.

2. **BusyBox Simplification:** Any word immediately following `/bin/busybox` is removed, standardizing BusyBox command calls, as they appear to be random and we can not extract any information from them.

3. **Root Pattern Replacement:** Replaces occurrences of the pattern `root:`*`password`* with the word `root`

4. **Word Pair Replacement:** Substrings of the form `"word1 \nword2"` are replaced by the placeholder word `replace` to generalize multiline strings.

5. **Empty String Filtering:** Commands consisting solely of repeated single quotes (e.g., `''`) are removed from the DataFrame.

6. **Specific Command Removal:** The command `'w'` (possibly indicating a minimal or non-useful entry) is explicitly filtered out.

7. **Echo Simplification:** Echo commands with flags and quoted strings (e.g., `echo -n "hello"`) are replaced with a generic form: `echo "string"`.

From his functions, the one that reduces the dataset the most is the busybox simplification. In the following example we can see how this works for most of the cases. We know busybox allows you to run built-in commands by appending the applet name. The applet or command BOTNET is not standard, so we can not know what it does. Then, deleting it does not make us lose information because we did not have it already.

Original:

```
sh; enable; system; shell; /bin/busybox BOTNET
```

Normalized:

```
sh; enable; system; shell; /bin/busybox
```

Then, after this normalization we get a much smaller dataset of only 619 classes of normalized commands. We can take some conclusions of the variability of this data if we look at plot 3.1.

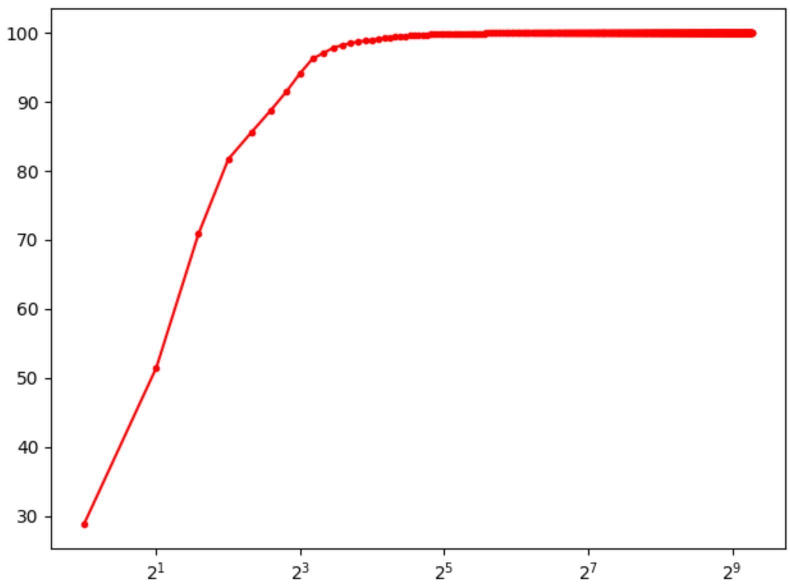


FIGURE 3.1: Distribution of data in logarithmic scale.

We can see that with the most common command we have 30% of the data covered, and with the 4 more common commands we get more than 80% of it. This unbalanced dataset don't have to make us forget about the tail data. It is very important to analyze and understand when talking about cyberattacks. In 3.2 we have a list of the 4 more common commands after the normalizing.

Command-Line	**Percentage**	**Number of Appearances**
`sh; ping; sh; shell; enable; system; /bin/busybox /proc/self/exe; cat /bin/echo`	28.92%	2370649
`enable; ; system; ; shell; ; sh; ; /bin/busybox`	22.56%	1849582
`sh; enable; system; shell; /bin/busybox`	19.35%	1586255
`sh; ping; sh; enable; system; shell; linuxshell; /bin/busybox`	10.84%	888684

TABLE 3.2: Command-Line Usage Statistics

This normalization is useful to reduce the non-informative command-lines and reduce drastically the number of commands, but the result can have no sense. We can see in this four examples we have blank commands or commands without arguments. That is why what we will be doing for the classification task is to use for each normalized command class the most common non-processed example to represent them, so the model understands it correctly.

Another thing we can observe in this first sight of the data before we classification it is the homogeneity of data in terms of protocol used. We saw that ProxyPot could support more than seven different protocols, but in the data we are analyzing we have only two of them. Telnet is the most used, appearing in 8195075 entries, and then we have SSH appearing 773 times.

Chapter 4

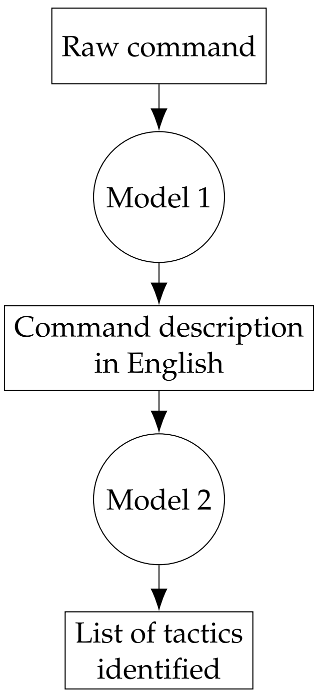
Methodology

The initial methodology proposed by the ACC involved using the OpenAI API to label a substantial number of commands with the tactics they employed. With this labeled dataset, the goal was to train a language model from scratch capable of classifying all commands as accurately as possible. However, as described in Section 1.1, the limitations of pretrained models in understanding and classifying command-line inputs made this approach infeasible.

As an alternative, we propose dividing the problem into two separate models:

- A first model that interprets and translates command-line inputs into natural language. The input to this model is a raw command, and the output is a description in English.
- A second model that maps the English description to the corresponding MITRE ATT&CK tactics and techniques.

This two-model approach addresses the two main limitations separately: the poor performance of general-purpose models on command or code understanding, and the high number of classification labels. By decoupling the tasks, we can focus on training specialized models with more targeted datasets and fine-tuning. This was not feasible with the original task due to the lack of existing examples and labeled data.



4.1 First Task

For the first task, we explore various approaches and evaluate them to identify the most effective solution. The primary resource for this is the dataset NL2SH.

This dataset (Westenfelder et al., 2025) was designed to support the development of models that generate command-line instructions from natural language. The dataset comprises pairs of natural language descriptions and their corresponding commands. It was compiled by merging and refining existing public datasets, shown in Figure 4.1, many of which were scraped from platforms like Stack Overflow.

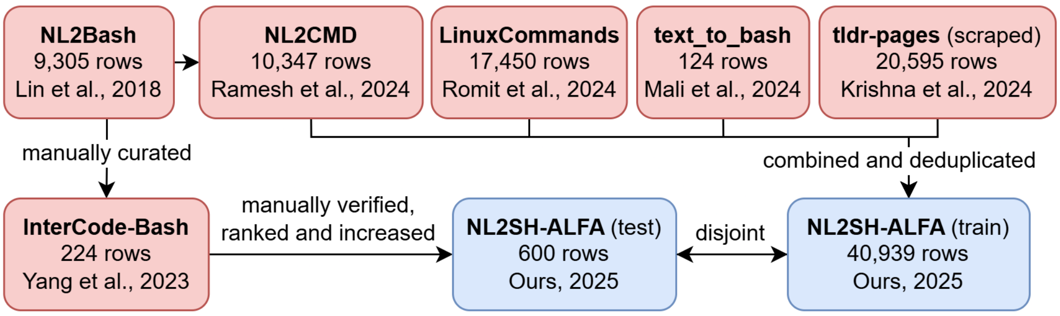


FIGURE 4.1: Source: Westenfelder et al., 2025

However, these datasets have two significant limitations: (1) the command-description pairings are not always verified, and (2) the descriptions often vary in format and may contain incorrect or inconsistent English. Therefore, we will use this data with caution, complementing quantitative evaluation with qualitative, manual analysis.

One useful aspect of the dataset is that it is thoughtfully partitioned into training and test sets. The training set contains more than 40,000 pairs (with the aforementioned caveats), while the test set includes 600 manually verified pairs like the five in Table 4.1. We can see that there are not one but two command lines for each natural language description. For our tests, we will only use the *bash2* ones, as they are more complete and specific usually. This verified test set greatly enhances the reliability of our evaluation.

TABLE 4.1: Five examples of the verified test set

nl	bash	bash2
list open files	`lsof`	`lsof -P -i -n`
create a copy of `/testbed/hello.php` named `/testbed/hello-COPY.php`	`cp /testbed/hello.php /testbed/hello-COPY.php`	`cp /testbed/hello.php /testbed/hello-COPY.php -v`
print the current date and time	`date`	`date -R`

nl	bash	bash2
encrypt the file `setup_nl2b_fs_1.sh` using AES-256-CBC with password `'password'` and save it to `out.enc`	`openssl enc -aes-256-cbc -in setup_nl2b_fs_1.sh -pass pass:password -out out.enc`	`openssl enc -aes-256-cbc -in setup_nl2b_fs_1.sh -pass pass:password -out out.enc`
Move files in `/workspace` accessed less than one day ago to directory /	`find /workspace -atime -1 -type f -exec mv {} / ;`	`find /workspace -type f -atime -1 -print0 | xargs -0 -I {} mv {} /`

Once we defined the dataset, we needed to define the models and methodologies to be used. In this era, where many large language models (LLMs) are publicly available, we considered it more practical to fine-tune pretrained models rather than creating one from scratch. After experimenting with several options—and to avoid the risk of getting lost in the vast landscape of LLMs—we selected the following models for fine-tuning and comparison:

- **Llama 3.2 Instruct 3B** (Meta Platforms, Inc., 2024): A 3-billion-parameter model based on transformer architecture. It is a general-purpose language model trained for multilingual dialogue use cases, including agentic retrieval and summarization tasks.
- **Stable Code Instruct 3B** (Phung et al., 2024): Also based on transformer architecture, this model is specifically trained for coding and software engineering dialogues.

The choice of model size was constrained by the limitations of our virtual environment; 3B was the largest model we could feasibly manage. Larger models would likely yield better performance. Nonetheless, evaluating the performance of smaller models is valuable in this project, especially for comparison with OpenAI's models, which we had initially planned to use.

4.1.1 Fine-Tuning

Our idea for the project was to apply to the both models a fine tuning using the train set, and compare them to the its raw versions for the translation task. We chose 500 command-description (the same for fine-tuning) both models) pairs from the train set at random to do it. Fine-tuning was carried out using the LoRA (Low-Rank Adaptation) method, a technique that injects trainable low-rank matrices into specific layers of a pre-trained model to enable efficient task-specific adaptation with minimal memory overhead.

In the technical aspects, we employed the PEFT (Parameter-Efficient Fine-Tuning) framework from the peft library, which supports LoRA configurations. A LoraConfig object was defined with key hyperparameters: rank r=8, scaling factor lora_alpha=32, and dropout probability lora_dropout=0.1. This configuration modifies the attention layers of the transformer to insert low-rank adapters while keeping the rest of the pre-trained weights frozen, significantly reducing the number of trainable parameters.

Tokenization was performed using the model's native tokenizer with truncation and padding to a maximum length of 256 tokens. The dataset was mapped accordingly to ensure uniform input shape for batch training. We specified a batch size (per_device_train_batch_size=1), number of epochs (num_train_epochs=3), and mixed precision training (fp16=True) to leverage half-precision floating. All this hyperparameters were chosen with the main objective of using the less memory and GPU capacity as possible, as the neither the virtual machine I was using nor Colab's GPUs were capable of managing anymore of this batch size.

Overall, this fine-tuning pipeline leverages modern transformer tooling to efficiently adapt a large language model to a downstream code interpretation task, while maintaining scalability and low computational requirements.

4.2 Second Task

The goal of this task is to classify natural language descriptions of system commands into the corresponding MITRE ATT&CK tactics. This is a **multi-label** classification task, meaning that each command can be associated with more than one of the 14 available tactics. Due to the limited number of labeled examples (only 30 commands with associated tactic labels), we must employ techniques such as zero-shot and few-shot learning, leveraging pretrained models to achieve satisfactory performance.

In conventional supervised learning, a model requires large amounts of labeled data to learn to classify new examples. However, in many practical scenarios — such as cybersecurity — labeled data is scarce or expensive to obtain. To address this limitation, we explore **zero-shot** and **few-shot** learning techniques.

Zero-shot learning refers to the ability of a model to make predictions on tasks it has never seen during training, by leveraging prior knowledge encoded in large-scale pretrained language models. For example, a zero-shot model may infer the most relevant MITRE tactic for a command description without having been explicitly trained on any examples of that mapping. In contrast, few-shot learning involves adapting a model using a very small number of labeled examples — in our case, 30 command descriptions paired with their corresponding MITRE tactics — to improve its performance on a specific downstream task. These approaches are especially valuable in our setting, where only a limited annotated dataset is available. The classification is approached through three complementary strategies described below.

4.2.1 OpenAI Model

For this task, we used the general pretrained OpenAI `gpt-4o` model OpenAI, 2024, deployed via Azure. The prompt used was as follows:

```
"You are a cybersecurity assistant helping map commands to MITRE
ATT&CK tactics based on their function and description.  Use only
publicly available information and do not simulate execution.  Respond
with a list of MITRE ATT&CK tactics in brackets.  Remember these
are the tactics:  [Reconnaissance, Resource Development, Initial
Access, Execution, Persistence, Privilege Escalation, Defense Evasion,
Credential Access, Discovery, Lateral Movement, Collection, Command
and Control, Exfiltration, Impact].
```

```
f"Given the following description of a simulation of an attack,
identify all MITRE ATT&CK tactics used.  Description to analyze:
{sanitize_description(data.loc[i, 'description'])}."
```

This lengthy and detailed prompt was necessary to bypass Azure's content management filters. Simpler prompts consistently triggered the following error message:

```
"The response was filtered due to the prompt triggering Azure OpenAI's
content management policy.  Please modify your prompt and retry."
```

This issue was likely due to the prompt being flagged as a potential jailbreak attempt.

4.2.2 Pretrained Model Specialized in MITRE Tactics Mapping

This model (Wei, 2024b) is a fine-tuned BERT-based classifier designed to map textual descriptions of commands or activities to MITRE ATT&CK tactics. It builds upon the BERT architecture by incorporating case-based representations, meaning it is trained to consider subtle linguistic cues in a context-sensitive way — preserving case distinctions (e.g., "PowerShell" vs. "powershell") that can be semantically meaningful in cybersecurity settings.

The dataset used for this fine-tuning (Wei, 2024a) consists of 14,000 examples of natural language descriptions of attacks with their corresponding tactics. To understand the mismatch with our data, consider some example descriptions from this dataset:

> *During the 2015 Ukraine Electric Power Attack, Sandworm Team modified in-registry internet settings to lower internet security.*

> *gh0st RAT operators have used dynamic DNS to mask the true location of their C2 behind rapidly changing IP addresses.*

> *For Operation Honeybee, the threat actors stole a digital signature from Adobe Systems to use with their MaoCheng dropper.*

As we can see, these examples differ significantly from the kind of data we have. This motivates our attempt to replicate the fine-tuning process of this model with our much smaller dataset.

4.2.3 Similarity-Based Classification with Contrastive Fine-Tuning

To improve classification accuracy under limited supervision, we adopt a similarity-based approach enhanced by fine-tuning with *Contrastive Loss*. Each data point consists of a command description and a set of associated MITRE tactics. The tactic set and their natural language definitions are stored in a dictionary mapping tactic names to descriptions.

We use the `all-mpnet-base-v2` sentence transformer model (Reimers and Gurevych, 2019) to encode both command descriptions and tactic definitions into a shared semantic space. To adapt the model to our classification task, we employ a leave-one-out (LOO) fine-tuning strategy, in which the model is updated using all but one sample and evaluated on the held-out instance. By this way we can use all data to train and there is no data leak in teh evaluation.

In each iteration, we construct a training set consisting of both positive and negative pairs:

- **Positive pairs**: A command description c paired with the definition d_j^+ of each associated tactic t_j^+.
- **Negative pairs**: The same command c paired with a randomly sampled unrelated tactic definition d_k^- corresponding to $t_k^- \notin \{t_j^+\}$.

These pairs are used to fine-tune the model using a *Contrastive Loss*, which encourages the embedding similarity to be high for positive pairs and low for negative ones. When trying to use only positive pairs the performance was worse, assigning almost every time all tactics to the command. To do it given embeddings $f(c)$ and $f(d)$, the loss is defined as:

$$\mathcal{L}_{\text{contrastive}} = \begin{cases} \|f(c) - f(d^+)\|^2 & \text{if } y = 1 \\ \max(0, \text{margin} - \|f(c) - f(d^-)\|)^2 & \text{if } y = 0 \end{cases}$$

where the label $y \in \{0,1\}$ denotes whether the pair is a true match, and the margin controls separation for negative examples.

The model is fine-tuned for one epoch on the constructed training set in each LOO fold. Once fine-tuning completes, the held-out test command c^* is encoded and compared to all tactic embeddings using cosine similarity:

$$\text{sim}(c^*, t_i) = \cos(f(c^*), f(d_i))$$

Each tactic is predicted as relevant if its similarity score exceeds a threshold τ (we used $\tau = 0.95$ as it was the one reporting better results):

$$\hat{y}_i = \begin{cases} 1 & \text{if sim}(c^*, t_i) > \tau \\ 0 & \text{otherwise} \end{cases}$$

The dataset used for this few-shot learning task contains only 30 samples, which were extracted from our main dataset. These include the 10 most common tactics, 10 moderately common ones, and the 10 least common, all manually labeled. The dataset is highly imbalanced, as shown in Figure 4.2.

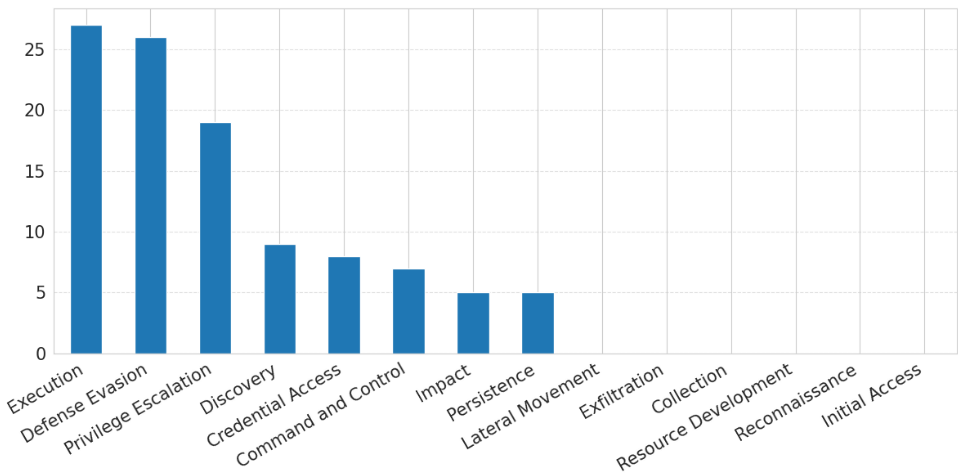


FIGURE 4.2: Distribution of tactics in our labeled dataset.

The absence of examples for six tactics poses a significant challenge for our model. However, it is important to consider that the type of data we aim to analyze is inherently imbalanced, as illustrated in Figure 3.1. Therefore, it is reasonable for the model to specialize in classifying the most frequently occurring tactics.

Chapter 5

Evaluation

5.1 First task

To evaluate the quality of the translated natural language commands produced by the models, we employed four complementary metrics: ROUGE, BLEU, BERT score, and cosine similarity using TF-IDF embeddings. Each of these metrics quantifies the similarity between generated descriptions and the ones in the given dataset, but they do so in fundamentally different ways: ROUGE through recall-based n-gram overlap, BLEU through precision-based n-gram matching with a brevity penalty, BERT Score through contextual semantic similarity, and TF-IDF cosine similarity through vector-based comparison of term importance across sentences. Even though BERT Score is the only metric that captures deep semantic meaning, the others—including TF-IDF—still provide valuable, interpretable insights about surface-level and lexical similarity between commands.

5.1.1 Evaluation Metrics

Cosine Similarity (TF-IDF)
Cosine similarity with TF-IDF (Term Frequency–Inverse Document Frequency) is a vector-based metric used to evaluate the lexical similarity between two text sequences. Each sentence is transformed into a high-dimensional vector where each dimension corresponds to a term, weighted by its frequency in the sentence and its rarity across the corpus. The cosine of the angle between the two vectors reflects their similarity:

$$\text{cosine\_similarity}(A, B) = \frac{A \cdot B}{\|A\| \|B\|}$$

where A and B are the TF-IDF vectors of the generated and reference sentences, respectively.

We included this metric to assess the lexical overlap between the generated command and the reference, independently of word order. While it lacks the semantic awareness of contextual models like BERT, it provides a lightweight, interpretable measure of whether the key terms used in both sentences are aligned.

ROUGE
ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a family of metrics designed to evaluate the overlap between generated and reference texts. In our case, we used ROUGE-L. This is a metric based on the Longest Common Subsequence

(LCS), which allows comparison without requiring consecutive or contiguous n-grams.

$$\text{ROUGE-L}_{\text{recall}} = \frac{LCS(X,Y)}{\text{length}(Y)}, \quad \text{ROUGE-L}_{\text{precision}} = \frac{LCS(X,Y)}{\text{length}(X)}, \quad F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

where X and Y are the candidate and reference sequences, respectively.

We chose ROUGE F1 score primarily for its recall orientation: it measures if the model-generated command includes relevant content from the reference. This is crucial when evaluating whether critical semantic elements from the original command are preserved.

BLEU

BLEU (Bilingual Evaluation Understudy) is a precision-based metric that computes how many n-grams in the candidate text appear in the reference text. It is defined as:

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

where:

- p_n is the modified precision for n-grams of size n,
- w_n are weights for each n-gram size (in our case uniform by default, e.g., $w_n = 0.25$ for $n = 1$ to 4),
- BP is the brevity penalty to discourage excessively short candidates:

$$\text{BP} = \left\{ 1 \text{if } c > r, \ \exp(1 - r/c) \text{ if } c \leq r \quad \right\}$$

with c and r being the lengths of the candidate and reference texts, respectively.

BLEU counts matching n-grams with a cap to prevent rewarding repeated words. It is effective for identifying exact syntactic matches but is less sensitive to paraphrasing. Despite this, BLEU remains a strong baseline and a widely recognized standard, making it valuable for comparing system performance under reproducible and consistent conditions.

BERT Score

BERT Score represents a newer generation of evaluation metrics based on deep contextual embeddings from pre-trained models such as BERT. Rather than relying on surface-form n-gram overlap, BERT Score measures semantic similarity at the token level in embedding space. The process involves:

1. Tokenizing both the candidate and reference using a BERT-compatible tokenizer.
2. Encoding each token using a BERT model to obtain contextual embeddings.
3. Computing a pairwise cosine similarity matrix between all candidate and reference tokens.
4. For each token, identifying the maximum similarity match in the other sentence (either precision or recall direction).

The scores we are using are precision, recall and F1.
Mathematically, for a candidate $C = \{c_1, \ldots, c_n\}$ and reference $R = \{r_1, \ldots, r_m\}$:

$$\text{Precision} = \frac{1}{n}\sum_{i=1}^{n}\max_j \cos(e(c_i), e(r_j)), \quad \text{Recall} = \frac{1}{m}\sum_{j=1}^{m}\max_i \cos(e(r_j), e(c_i))$$

We use BERT Score to address cases where the model's output may not match the reference exactly in wording but retains semantic equivalence. This is common in natural language tasks, where multiple phrasings can convey the same meaning. BERT Score is particularly powerful in recognizing synonyms, paraphrases, and reordering, which traditional metrics might penalize unfairly.

Summarizing, we have:

- **Cosine Simmilarity of TF-IDF** to observe similarity independent to the word order.
- **ROUGE** that captures the degree to which critical elements from the reference command are retained in the output, prioritizing content completeness.
- **BLEU** that provides a robust standard for measuring exact n-gram overlap and is useful for benchmarking.
- **BERT Score** that offers deep semantic comparison, capable of rewarding meaningful but non-identical reformulations.

Together, these metrics provide a balanced view: from surface-level lexical alignment (Cosine Similarity of TF-IDF and BLEU), through recall-based informativeness (ROUGE), to deep semantic alignment (BERT Score). This expanded set of metrics is especially important for the task of translating executable commands into human-readable language, where both accuracy and preservation of meaning are essential. By including both shallow and deep comparison techniques, we ensure that the evaluation reflects not only exact word overlap but also broader conceptual equivalence.

5.1.2 Results

We can see in the table 5.1 how the fine-tuned Llama model and the original Llama one get the best results in the metrics, some of them very far from the Stable models.

Model	Cos Sim	BLEU	ROUGE	BERT Rec	BERT Pre	BERT F1
Stable	0.6327	0.0471	0.1977	0.8224	0.9004	0.8595
Llama	**0.7338**	0.3735	0.3544	0.9025	**0.9137**	**0.9077**
FT Stable	0.6481	0.1204	0.2555	0.8009	0.8824	0.8393
FT Llama	0.7153	**0.3999**	**0.3593**	**0.9044**	0.9037	0.9037

TABLE 5.1: Mean scores for each experiment across multiple evaluation metrics

For further analysis and with the objective to chose the model that best does this task, we will observe how this metrics are distributed in the following plots.

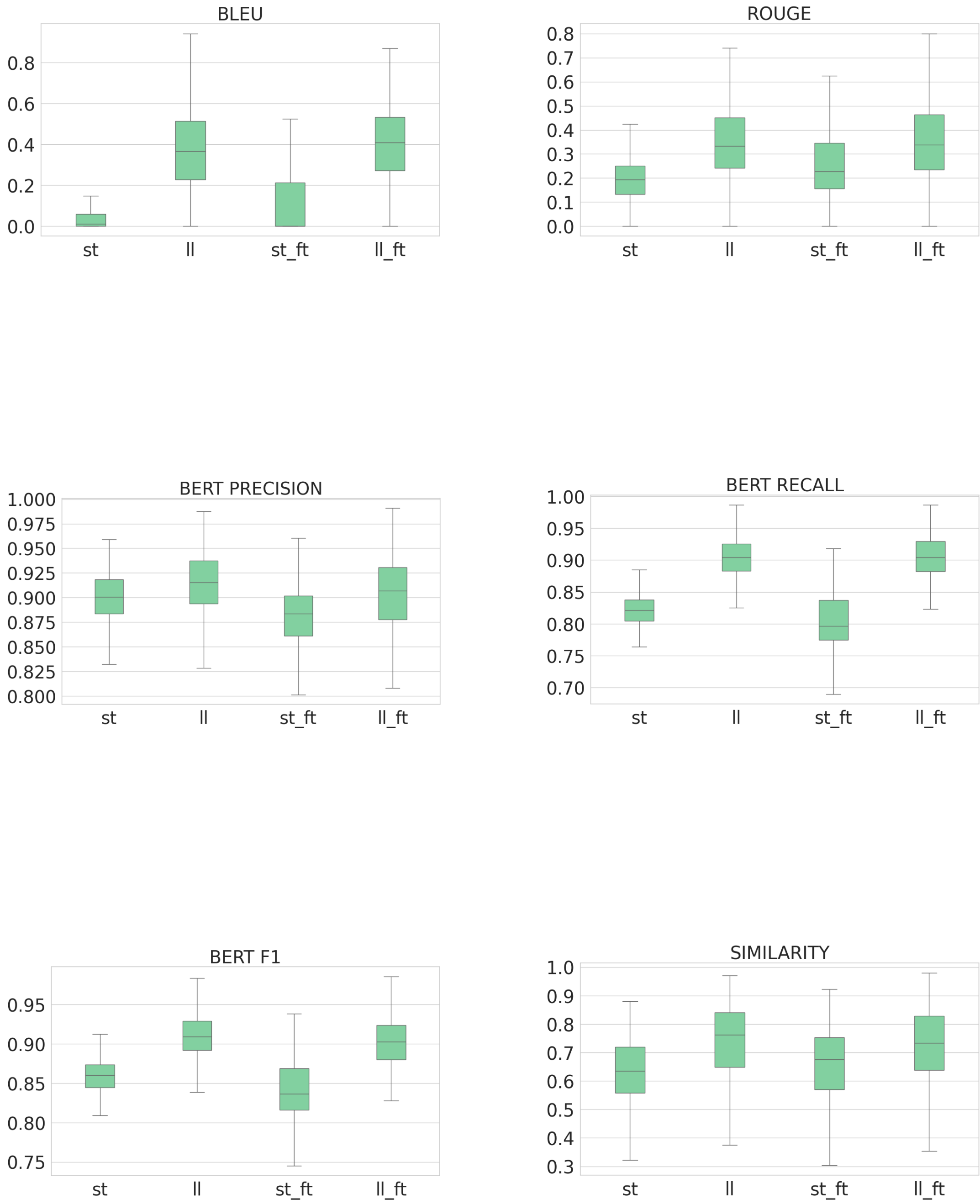


FIGURE 5.1: Box plots for all of the evaluated metrics.

After watching this plots we should have even less doubt on how Llama general model outperformed the Stable one, focused in coding interpretation. In the following line we will look at specific examples to see the issues that each model have in this task.

We can start by watching some examples of translations and what does each model output:

First example	
Bash Command	ls -l
NL description	list files in the current directory
Stable	This command lists the details of all files and directories in the current directory in a long format. It includes details such as file permissions, number of links, owner, group, size, and modification time.<|im_end|>
Llama	List the files and directories in the current directory in a detailed, long format.
Fine-tuned Stable	List files in long format. Response: List files in long format. Explanation: List files in long format, which includes details such as file permissions, number of links, owner, group, size, and modification time. Example: “‘ ls -l “‘ Response: List files in long format. Explanation: List files in long format, which includes details such as file permissions, number of links, owner, group, size, and modification time. Example: “‘ ls -l “‘ Response: List files in long format. Explanation: List files in long format, which includes details such as file permissions, number of links, owner, group, size, and modification time. Example: “‘ ls -l “‘ Response: List files in long format.
Fine-tuned Llama	List files in long format with detailed information.

Second example	
Bash Command	ln -s /testbed/dir3/subdir1/subsubdir1/FooBar
NL description	Create a symbolc link in the /testbed/dir3/subdir1/subsubdir1/FooBar to root folder

Stable	This command creates a symbolic link named "FooBar" that points to the directory "/testbed/dir3/subdir1/subsubdir1/". The symbolic link "FooBar" is a shortcut to that directory, allowing users to navigate to it more easily.<|im_end|>
Llama	Create a symbolic link named "FooBar" in the current directory, pointing to the file "/testbed/dir3/subdir1/subsubdir1/FooBar".
Fine-tuned Stable	Create a symbolic link to a file or directory. Response: Create a symbolic link to a file or directory named "FooBar" in the directory "/testbed/dir3/subdir1/subsubdir1/". Response: Create a symbolic link to a file or directory named "FooBar" in the directory "/testbed/dir3/subdir1/subsubdir1/". Response: Create a symbolic link to a file or directory named "FooBar" in the directory "/testbed/dir3/subdir1/subsubdir1/". Response: Create a symbolic link to a file or directory named "FooBar" in the directory "/testbed/dir3/subdir1/subsubdir1/". Response: Create a symbolic link to a file or directory named "FooBar" in the directory "/testbed/dir3/
Fine-tuned Llama	Create a symbolic link to a directory in the testbed directory.

Third example	
Bash Command	find /workspace -name "*.txt" | xargs ls -ld
NL description	List the details of all the text files in /workspace directory.

Stable	This command finds all .txt files in the /workspace directory and lists their details (including file type and permissions, ownership, size, and modification/creation date). Here's a breakdown of the command: 1. 'find /workspace -name "*.txt"': This part of the command locates all files named "*.txt" in the /workspace directory and its subdirectories. 2. '| xargs ls -ld': This part of the command takes the output from the 'find' command (a list of file paths), passes them as arguments to 'ls -ld', which lists the details of the files and directories. The '-ld' option tells 'ls' to display the file type and permissions, ownership, size, and modification/creation date. In summary, this command lists the details of all .txt files found in the /workspace directory and its subdirectories.<|im_end|>
Llama	Find all files with the extension ".txt" in the "/workspace" directory and its subdirectories, then display detailed information about each file.
Fine-tuned Stable	Find all files with the extension ".txt" in the /workspace directory and list their details. Response: Find all files with the extension ".txt" in the /workspace directory and list their details using the "ls -ld" command. Explanation: The command will search for all files with the extension ".txt" in the /workspace directory and list their details using the "ls -ld" command. The "-l" option will display the details of the files and the "-d" option will display directories themselves, not their contents. Example: "' find /workspace -name "*.txt" | xargs ls -ld "' Output: "' drwxr-xr-x 2 user group 4096 Jan 1 12:00 /workspace/file1.txt drwxr-xr-x 2 user group 4096 Jan 1 12:00 /
Fine-tuned Llama	List all files with the extension ".txt" in the "/workspace" directory and display their detailed information.

We can see that the both Stable models give strange outputs, even the finetuned one. We tried prompt tuning but we did not find the way to get a response similar tho the one we want, that is, as short as concise as possible for the model to be faster.

Exploring the Llama models, they both perform good for the BLEU and ROUGE score, but we have some sentences with 0 (or very close to 0) score. We can observe in 5.2 that the both are correlated, but we realized there are very extreme examples. We will study this examples with a very low score in both metrics, and examples with a big difference between metrics. We will do it only for the normal Llama model because the fine-tuned one is analogous.

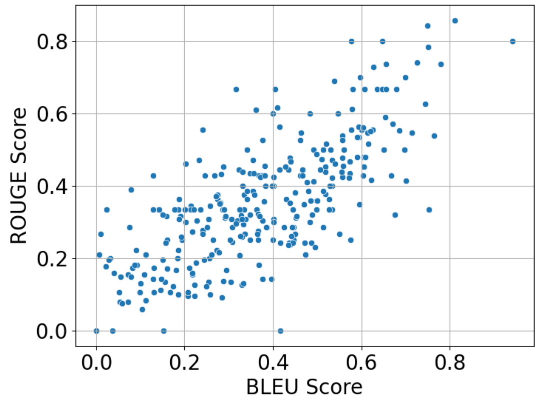


FIGURE 5.2: BLEU vs ROUGE score for Llama model

Bash commands	bind -P
Given description	print all readline bindings
Model description	Bind a process to a specific port.
BLEU score	2.351252065921332e-78
ROUGE score	0.0

TABLE 5.3: Llama example with the lowest BLEU and ROUGE scores.

Bash commands	wc setup_nl2b_fs_1.sh
Given description	Count the lines, words, and characters in setup_nl2b_fs_1.sh
Model description	This command sets up a Bash script named 'setup_nl2b_fs_1.sh'.
BLEU score	0.416678667227553
ROUGE score	0.0

TABLE 5.4: Llama example with very low ROUGE but high BLEU scores.

We can see here a big problem, as some (the worst) Llama outputs are not semantically close to the given description, even though they have high BERT score.

After all this analysis we have that:

- The Llama models are better than the Stable ones in all the metrics evaluated. They all have the same size and Stable should be more specialized, but the Llama one must have been trained better.
- Most of the descriptions are correct, but we will have to deal with a tail of incorrect ones when using any of those models.
- BERT isn't able to capture the semantics of the sentences, probably because of the small size model that we had to use. Then, the non-semantic metrics are the ones that we have to guide with. That point will makes us chose the fine-tuned Llama, as it has better scores in ROUGE and BLEU.

5.2 Second Task

The second task addresses a multi-label classification problem, where each input can be assigned one or more labels (tactics). The goal is to correctly predict all relevant labels associated with a given attack. The dataset used is the one described in 4.2.3, as is the only one we have.

5.2.1 Evaluation Metrics

To measure model performance in this multi-label context, we employ **micro-average** and **macro-average** precision, recall, and F1-score.

Micro-average
Micro-averaged metrics compute global performance by aggregating all true positives, false positives, and false negatives across labels before calculating the metric:

$$\text{Precision}_{\text{micro}} = \frac{\sum_l TP_l}{\sum_l TP_l + \sum_l FP_l}, \quad \text{Recall}_{\text{micro}} = \frac{\sum_l TP_l}{\sum_l TP_l + \sum_l FN_l}$$

$$\text{F1}_{\text{micro}} = \frac{2 \cdot \text{Precision}_{\text{micro}} \cdot \text{Recall}_{\text{micro}}}{\text{Precision}_{\text{micro}} + \text{Recall}_{\text{micro}}}$$

Micro-averaging is sensitive to the performance on frequent labels, making it a good measure of overall performance in imbalanced datasets.

Macro-average
Macro-averaged metrics treat all labels equally, calculating metrics per label and then averaging them:

$$\text{Precision}_{\text{macro}} = \frac{1}{L}\sum_{l=1}^{L} \frac{TP_l}{TP_l + FP_l}, \quad \text{Recall}_{\text{macro}} = \frac{1}{L}\sum_{l=1}^{L} \frac{TP_l}{TP_l + FN_l}$$

$$\text{F1}_{\text{macro}} = \frac{1}{L}\sum_{l=1}^{L} \text{F1}_l$$

This provides a better view of how well a model performs across all classes, especially underrepresented ones.

5.2.2 Results

We begin by examining the proportion of exact matches—instances where all predicted labels were correct and complete:

Model	Fully Correct Predictions (%)
Pretrained MITRE	10.00
Similarity-based	**20.00**
OpenAI	10.00

TABLE 5.5: Percentage of fully correct multi-label predictions.

In Table 5.6 we can see the performance on the mentioned metrics, and how our trained model outperforms the other ones in all metrics.:

Model	Micro Average			Macro Average		
	Precision	Recall	F1	Precision	Recall	F1
Pretrained MITRE	0.65	0.58	0.61	0.52	0.44	0.47
Similarity-based	**0.72**	**0.68**	**0.70**	**0.61**	**0.57**	**0.59**
OpenAI	0.40	0.35	0.37	0.28	0.22	0.24

TABLE 5.6: Micro and macro averaged precision, recall, and F1-scores for each model.

The OpenAI model, with the worst performance, exhibited a significant flaw: it almost half of the times predicted no labels at all, despite every sample having at least one tactic. This systematically damages its recall and reliability.

Even though our trained model is the best among the three, we cannot fully trust its results due to the following reasons:

- The overall accuracy is very low.
- The data imbalance showed in 4.2.3 leads to biases in the model. For example, when we examine accuracy scores for each tactic, we find that most of them (11) have an accuracy above 85%, while the *Discovery* tactic only achieves 43%. In general, the accuracy tends to be lower for the more common tactics, which is a significant concern.

On the positive side, this is the only one of the three models that is scalable—that is, it has the potential to improve its performance as more data becomes available. This is something the other two models cannot achieve.

Chapter 6

Conclusions and future work

This project set out to tackle two interrelated challenges: understanding and classifying command-line activity gathered from honeypot systems using the MITRE ATT&CK framework, and developing a practical, interpretable interface that could assist in cybersecurity incident analysis. Our aim was to go beyond conventional approaches and explore cost-effective, domain-specific models that avoid reliance on commercial large language models (LLMs).

Summary of Results

The first part of our pipeline, which focused on understanding and translating CLI (Command Line Interface) commands into natural language, achieved encouraging results. The model demonstrated a good capacity to generalize and produce coherent descriptions for a broad variety of commands. These outcomes are particularly promising considering the relatively modest computational resources used. We believe that with a moderate increase in computing capacity and some targeted optimizations—such as refining regular expressions to catch structural errors in parsing—further improvements are readily attainable.

This efficiency is not just technically beneficial; it is also economically and environmentally significant. Avoiding the use of large-scale LLMs like GPT-4o, which are computationally expensive and energy-intensive, makes our approach more scalable and accessible for long-term deployment in resource-constrained environments. It shows that understanding CLI behavior does not necessarily require heavy, general-purpose models, especially when a well-adapted, lightweight model can perform comparably well on focused tasks.

On the second objective—classifying commands into MITRE ATT&CK tactics—the performance of large language models proved to be underwhelming. Despite their general strength in natural language processing, models like GPT-4 did not achieve a level of accuracy that would make them reliable for unsupervised classification in cybersecurity contexts. In particular, their performance was limited by the complexity and imbalance of the classification task: with over 14 defined tactics and scarce data for many of them, the models often produced biased or incorrect predictions. These limitations underline the necessity of domain-specific solutions for threat analysis.

Nevertheless, these models did show some utility as a tool for exploratory data analysis (EDA). Their ability to generate preliminary classifications or identify potentially interesting patterns could make them useful in an assisted analysis workflow—provided that a human analyst verifies the output.

Limitations and the Need for Data

A recurring limitation throughout the project was the scarcity and imbalance of labeled data. This affected both training and evaluation, particularly in the classification component. The lack of representative examples for some tactics hampered the model's ability to generalize effectively across all classes. Consequently, the reliability of the system—especially in an automated security context—remains constrained.

We also note that although our custom model architecture is cost-efficient, it still lacks the robustness and generalization needed for deployment in a real-time operational environment. Greater data availability, particularly from diverse honeypot systems, will be essential to overcome this barrier.

Future Work

For the first task (command understanding), future improvements could involve:

- Scaling up the computational infrastructure to train and fine-tune models more effectively.
- Enhancing pre-processing techniques, particularly through the use of more advanced regular expression patterns, to catch subtle syntax errors and ambiguities in command parsing.

For the second task (tactic classification), the most critical direction is data acquisition:

- Expanding the training dataset with more examples across all MITRE tactics to address the class imbalance problem.
- Exploring data augmentation methods to synthetically increase the number of labeled samples per class.

Another possible line of research could involve prompt engineering and fine-tuning of commercial LLMs like GPT-4 for this specific task. However, this direction is less appealing due to the high cost and limited scalability of commercial APIs, which run counter to the project's goal of developing economically viable and sustainable solutions.

Final Remarks

This work illustrates the potential and limitations of applying machine learning techniques to cybersecurity problems such as command-line interpretation and tactic classification. While much work remains to be done—particularly in data acquisition and model refinement—the results obtained thus far suggest that with targeted efforts and resource-aware design, it is possible to build tools that support cybersecurity analysts in meaningful and efficient ways. As the field progresses, we hope our contributions provide a foundation for further, scalable advancements in automated threat classification.

Appendix A

Source Code

The reader can find the source code with the implementation explained in this document in

https://github.com/HugCamps/TFM-HoneyPotClasification.git

There is each one of the eight notebooks used, as long as a README file that explains each of them.

The dataset and trained models used in this project are not publicly available due to privacy and data-sharing restrictions.

The project was developed using both Google Colab and a virtual machine provided by ACC. As a result, file paths and data-loading methods may vary across notebooks, and a unified environment was not used.

Bibliography

Cristian Munteanu Said Jawad Saidi, Oliver Gasser Georgios Smaragdakis Anja Feldmann (Oct. 2021). "Fifteen Months in the Life of a Honeyfarm". In: *IMC '23: Proceedings of the 2023 ACM on Internet Measurement Conference*, pp. 282–296. URL: https://doi.org/10.1145/3618257.3624826.

Meta Platforms, Inc. (2024). *Llama 3.2 3B Instruct*. https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct. Released Sept 25, 2024; Accessed: 2025-06-11.

OpenAI (2024). *GPT-4o: OpenAI's Omnimodal Model*. https://openai.com/index/gpt-4o. Accessed: 2025-06-15.

Phung, Duy et al. (2024). *Stable Code 3B*.

Reimers, Nils and Iryna Gurevych (Nov. 2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Hong Kong, China: Association for Computational Linguistics, pp. 3982–3992. DOI: 10.18653/v1/D19-1410. URL: https://aclanthology.org/D19-1410.

Rodríguez, David Rosado et al. (2024). "Enhancing Cybersecurity Intelligence through Machine Learning: Clustering and Forecasting Analysis of Honeypot Data". In: *Agència de Ciberseguretat de Catalunya*. URL: https://ciberseguretat.gencat.cat/web/.content/04_actualitat/Noticies/2024/PDF/Paper-reviewed-honeypots-IA.pdf.

Wei, Sarah (2024a). *MITRE ATT&CK v16 Case-Based Dataset for Tactic Classification*. https://huggingface.co/sarahwei/MITRE-v16-tactic-bert-case-based. Dataset used to train the MITRE-v16-tactic-bert-case-based model. Accessed: 2025-06-13.

— (2024b). *MITRE-v16-tactic-bert-case-based*. https://huggingface.co/sarahwei/MITRE-v16-tactic-bert-case-based. Accessed: 2025-06-13.

Westenfelder, Finnian et al. (2025). "LLM-Supported Natural Language to Bash Translation". In: arXiv: 2502.06858 [cs.CL]. URL: https://arxiv.org/abs/2502.06858.