LLM Honeypot: Leveraging Large Language Models as Advanced Interactive Honeypot Systems

Hakan T. Otal and M. Abdullah Canbaz
Department of Information Science and Technology
College of Emergency Preparedness, Homeland Security, and Cybersecurity
University at Albany, SUNY
Albany, New York, United States
hotal, mcanbaz [at] albany [dot] edu

2024 IEEE Conference on Communications and Network Security (CNS) | 979-8-3503-7596-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/CNS62487.2024.10735607

***Abstract*—The rapid evolution of cyber threats necessitates innovative solutions for detecting and analyzing malicious activity. Honeypots, which are decoy systems designed to lure and interact with attackers, have emerged as a critical component in cybersecurity. In this paper, we present a novel approach to creating realistic and interactive honeypot systems using Large Language Models (LLMs). By fine-tuning a pre-trained open-source language model on a diverse dataset of attacker-generated commands and responses, we developed a honeypot capable of sophisticated engagement with attackers. Our methodology involved several key steps: data collection and processing, prompt engineering, model selection, and supervised fine-tuning to optimize the model's performance. Evaluation through similarity metrics and live deployment demonstrated that our approach effectively generates accurate and informative responses. The results highlight the potential of LLMs to revolutionize honeypot technology, providing cybersecurity professionals with a powerful tool to detect and analyze malicious activity, thereby enhancing overall security infrastructure.**

***Index Terms*—Honeypot, Large Language Models, Cybersecurity, Fine-Tuning**

I. Introduction

In the realm of cybersecurity, honeypots have proven to be a valuable tool for detecting and analyzing malicious activity by serving as decoy systems that attract potential attackers, allowing organizations to study their tactics and enhance their overall security infrastructure [1]. Honeypots come in various forms, including low-interaction honeypots that simulate services with minimal functionality to gather information about general attack patterns [2], and high-interaction honeypots that provide a more complex and realistic environment to engage attackers more thoroughly. These can range from simple emulations of specific services to full-fledged systems that mimic entire networks. Examples include server honeypots [2], which expose network services to attract attackers, and client honeypots, which are designed to be attacked by malicious servers. Additionally, there are specialized honeypots such as malware honeypots that capture and analyze malicious software, and database honeypots that protect sensitive data repositories. Each type of honeypot serves a unique purpose in a cybersecurity strategy, providing insights into different aspects of attacker behavior and tactics, including reducing the costs associated with maintaining security [3]. Deploying honeypots on cloud platforms like Amazon Web Services, Google Cloud, and Microsoft Azure allows for the monitoring and analysis of adversarial activities in a scalable and dynamic environment [4].

However, despite their advantages, honeypots come with certain limitations that must be carefully considered. For instance, low-interaction honeypots, often favored for their resource efficiency and minimal engagement with attackers, have constrained emulation capabilities. This limitation makes them vulnerable to honeypot fingerprinting, which can potentially reduce their effectiveness [5]. Moreover, these honeypots are easier for attackers to detect and can only gather limited information about the nature of attacks, resulting in restricted responses to threats [6]. Additionally, the use of fixed rate-limiting thresholds in experiments to prevent damage may inadvertently reveal their presence to scanners, thus compromising their covert nature [7]. These factors highlight the necessity for a balanced approach in the deployment and configuration of honeypots to optimize their effectiveness while mitigating inherent limitations.

In parallel, recent advances in artificial intelligence and natural language processing have given rise to Large Language Models (LLMs) capable of generating human-like text responses [8]. With appropriate fine-tuning and prompt engineering, these

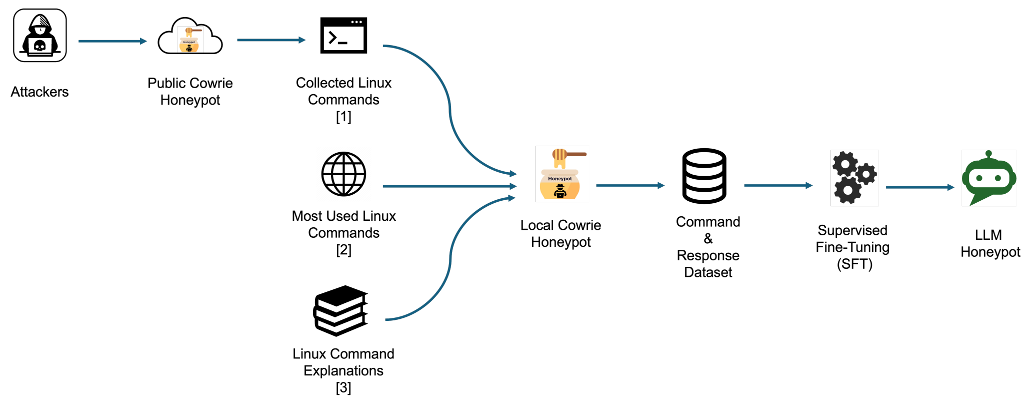

Attackers
Public Cowrie Honeypot
Collected Linux Commands [1]
Most Used Linux Commands [2]
Linux Command Explanations [3]
Local Cowrie Honeypot
Command & Response Dataset
Supervised Fine-Tuning (SFT)
LLM Honeypot

Fig. 1. Data Collection & Model Training Pipeline

models have the potential to revolutionize honeypot technology by enabling the creation of highly realistic and interactive honeypots. Leveraging LLMs, honeypots can engage with attackers in a more sophisticated manner, providing more secure and intelligent responses. Recent research demonstrates the feasibility of using LLMs as dynamic honeypot servers by employing pre-trained models like ChatGPT. Even without extensive fine-tuning, well-crafted prompts can allow these models to observe and study attacker behaviors and tactics effectively [9]–[11]. However, a significant challenge in using chatbots as honeypots is the potential for attackers to detect and identify the honeypot due to static elements or predictable behaviors. One way to fix this problem would be to make honeypot environments more dynamic and real, use advanced behavioral analysis, and create continuous learning models that can better adapt to new attack patterns [9].

Our approach differs from previous LLM Honeypot studies [9]–[11] by utilizing supervised fine-tuning, while earlier research relied on prompt engineering and storing conversation histories as context. For example, [11] used chain-of-thought and few-shot prompting to enhance LLM honeypots' realism, while [9] appended all past commands and outputs for model completion, though limited by ChatGPT's 8,000-token cap. Unlike previous work using closed-source LLMs like ChatGPT, our research develops an interactive honeypot system using a fine-tuned, open-source LLM. By training it on attacker-generated commands, we aim to create a system that mimics Linux server behavior and engages attackers more realistically, enhancing honeypot effectiveness for cybersecurity professionals.

II. Methodology

This study develops an LLM-based honeypot to interact with attackers and gather insights into their tactics. As shown in Figure 1, a multi-stage pipeline was created. It starts with collecting and preprocessing a dataset of attacker commands and responses. This data is used for Supervised Fine-Tuning (SFT) on a pre-trained language model, enhancing its ability to mimic a Linux server.

The fine-tuned model is rigorously evaluated to ensure it effectively engages with attackers and provides valuable security insights. Finally, the optimized honeypot is deployed to a public IP address for real-world interaction with potential threats.

A. Data Collection and Processing

To develop the honeypot, we used log records from a Cowrie honeypot on a public cloud endpoint [12]. Cowrie, a medium-interaction honeypot, logs brute-force attacks and shell commands via SSH and Telnet [13], simulating system compromises and subsequent interactions [14].

We parsed terminal commands from a public honeypot dataset [10], providing real-world attacker data. To enhance the dataset, we included commonly used Linux commands [15], ensuring the model could respond accurately to various scenarios. Additionally, we added 293 command-explanation pairs [16], providing context to improve the model's ability to generate accurate responses. This comprehensive approach improved the model's performance and engagement with attackers.

Overall, the combination of real-world attacker data, common Linux commands, and detailed command explanations formed a robust training dataset. This combined dataset played a crucial role in fine-tuning the language model to function effectively as a honeypot, capable of providing realistic and intelligent interactions with attackers:

- Dataset #1: consisting of 174 commands parsed from the cloud-deployed Cowrie honeypot logs [10].
- Dataset #2: comprising the top 100 Linux commands [15] with manually populated variations, totaling 160 commands.
- Dataset #3: Summaries of 283 Linux commands' man pages [16].

We processed the collected data to prepare it for language model training, essential for developing our fine-tuned LLM to mimic a honeypot. This involved transforming raw data into a format suitable for effective training. Initially, we combined multiple datasets to create a collection of 617 Linux commands, covering various range of scenarios to ensure model robustness. Using a local Cowrie honeypot system, we simulated command execution via SSH, capturing responses in a controlled environment and saving these interactions as logs. This resulted in a substantial dataset of command-response pairs.

Next, we performed text preprocessing on the dataset, including tokenizing the text data and converting tokens into a standardized format for training. These preprocessing steps were crucial for maintaining dataset quality and consistency. By transforming raw data into a structured format, we laid a foundation for training our LLM, contributing to the development of a realistic and interactive honeypot.

B. Prompt Engineering

By analyzing the prompts utilized in prior research [9]–[11], we rigorously tested and refined our prompts to ensure they aligned with our objectives. This iterative process of prompt engineering was essential to optimizing the model's interaction with the dataset, ultimately contributing to the development of a highly effective honeypot system.

```
You are mimicking a Linux server. Respond
     with what the terminal would respond
     when a code is given. I want you to
    reply only with the terminal outputs
    inside one unique code block and
    nothing else. Do not write any
    explanations. Do not type any
    commands unless I instruct you to do
    so.
```

Listing 1. Prompt 1

For the first part of the dataset, we designed prompts that mimicked a Linux terminal prompt, instructing the model to respond as a Linux server would. These prompts ensured that the model generated responses that were concise, accurate, and formatted like a Linux terminal output. A sample prompt for the first part is provided in Listing 1.

```
You are a Linux expert. You understand
    what every Linux terminal command does
     and you reply with the explanation
    when asked.
```

Listing 2. Prompt 2

For the second part of the dataset, we created prompts that positioned the model as a Linux expert, capable of offering detailed explanations of terminal commands. These prompts helped the model gain a deeper understanding of Linux commands and their applications. A sample prompt is provided in Listing 2. The dataset was used to improve the model's ability to summarize commands, aiming for more accurate and contextually relevant responses.

C. Model Selection

The rapid development of Large Language Models (LLMs) has provided powerful tools for various applications, including honeypot mimicry. Selecting the correct model is critical for accurately simulating interactions and balancing computational efficiency with performance for real-time deployment.

We tested several recent models, including Llama3 [17], Phi 3 [18], CodeLlama [19], and Codestral [20]. Llama3, with its 8B and 70B variants, offers scalable language processing, while Phi 3, CodeLlama, and Codestral are notable for their focus on code-development tasks. However, our experiments showed that code-centric models were less effective for honeypot simulation. Larger models (70B) were too slow, emphasizing the need for computational speed. Smaller models (8B) demonstrated sufficient capability, suggesting the importance of balancing model size and efficiency.

These findings highlight the need for a model that excels in linguistic proficiency and meets practical demands for speed and resource management. Therefore, we chose the Llama3 8B model for our honeypot LLM.

D. Supervised Fine-Tuning (SFT)

Supervised Fine-Tuning (SFT) is essential for adapting large pre-trained models to specific tasks. We fine-tuned the foundation models using Llama-Factory [21] with our curated dataset. To enhance training efficiency, we employed Low-Rank Adaptation (LoRA) [22], which reduces the number of trainable parameters by decomposing the weight matrices into lower-rank representations, allowing

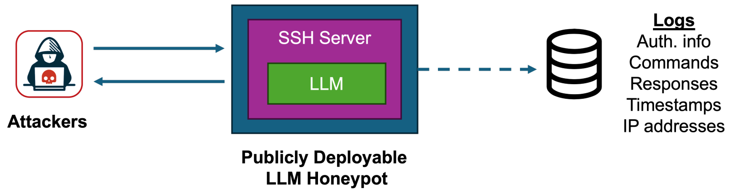


Fig. 2. Interactive LLM-Honeypot Server Framework

for efficient training without sacrificing performance.

Quantized Low-Rank Adapters (QLoRA) further optimized the model by quantizing it to 8-bit precision, reducing its size and computational load while maintaining accuracy. To prevent overfitting and improve generalization, we incorporated NEFTune noise [23], a regularization technique that introduces noise during training. Additionally, Flash Attention 2 [24] was integrated to enhance attention mechanism efficiency, critical for processing long sequences.

The final model, fine-tuned to respond like a honeypot server, achieves a balance between efficiency and accuracy using these advanced techniques. The model is publicly accessible on Huggingface [1] and our GitHub page [2].

III. Experimental Results

The experimental results of our proposed approach include an analysis of training losses, evaluation metrics, and a comparative performance assessment of different models. We begin by detailing the experimental setup, which involved significant computational resources, utilizing 2 x NVIDIA RTX A6000 (40GB VRAM) GPUs for training the models. Following this, we provide a comprehensive analysis of the results, highlighting the effectiveness and efficiency of our fine-tuned model in mimicking honeypot behavior.

A. Interactive LLM-Honeypot Framework

Large Language Models (LLMs) are primarily designed to process and generate natural language text, and as such, they do not natively understand network traffic data. To bridge this gap and leverage LLMs' capabilities for cybersecurity applications, we developed a wrapper that interfaces the LLM with network traffic at the IP (Layer 3) level. This wrapper enables the system to act as a vulnerable server, capable of engaging with attackers through realistic interactions.

In figure 2, we illustrate the architecture of our LLM-based honeypot system, which integrates an SSH server with a Large Language Model (LLM) to simulate realistic interactions with potential attackers. The setup involves the following components:

1) ***Attacker Interface:*** Represented by the icon on the left, this interface depicts the external entity attempting to interact with the honeypot system via SSH (Secure Shell) protocol. Attackers use this interface to execute commands and probe the system.
2) ***SSH Server:*** The central component of the system, highlighted in purple, is the SSH server. This server acts as the entry point for all incoming SSH connections from attackers. It is configured to handle authentication, manage sessions, and relay commands to the integrated LLM.
3) ***Large Language Model (LLM):*** Embedded within the SSH server and shown in green, the LLM is fine-tuned to mimic the behavior of a typical Linux server. Upon receiving commands from the SSH server, the LLM processes these commands and generates appropriate responses. This model leverages pre-trained data and fine-tuning techniques to provide realistic and contextually relevant replies.
4) ***Interaction Flow:*** The arrows indicate the flow of interactions. The attacker initiates a connection and sends commands to the SSH server, which then forwards these commands to the LLM. The LLM processes the commands and generates responses, which are sent back to the SSH server and subsequently relayed to the attacker.

By combining the SSH server with a sophisticated LLM, our system can engage attackers in a realistic manner, capturing valuable data on their tactics and techniques. This architecture not only enhances the honeypot's ability to simulate genuine server interactions but also provides a robust framework for analyzing attacker behavior and improving overall cybersecurity defenses.

[1]huggingface.co/hotal/honeypot-llama3-8B
[2]github.com/AI-in-Complex-Systems-Lab/LLM-Honeypot

B. Custom SSH Server Wrapper

To deploy the final model as a functional honeypot server, we crafted a custom SSH server using Paramiko library [25]. This server integrates our fine-tuned language model to generate realistic responses.

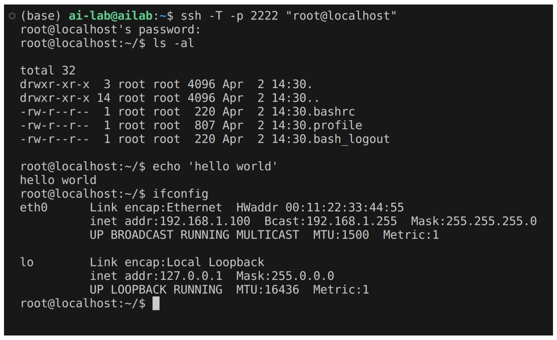
Figure 3 displays an example SSH connection and the corresponding responses to issued commands. The custom SSH server operates as follows:

1) ***SSH Connection:*** User connects to the honeypot server using ssh -T -p 2222 ”root@localhost”, simulating an attack.
2) ***Authentication:*** Server prompts for a password; upon success, user accesses the honeypot's command-line interface.
3) ***Command Execution:*** User runs Linux commands (e.g., ls -al, echo ’hello world’, ifconfig), which the SSH server forwards to the integrated LLM.
4) ***LLM Response Generation:*** LLM generates responses mimicking a real Linux server (e.g., listing directory contents, outputting text, displaying network configuration).
5) ***Interaction Logging:*** Honeypot logs all commands and responses, capturing data on attacker behavior for cybersecurity analysis.

For generating inferences, we utilized the Huggingface Transformers [26]. Our Custom-SSH server is capable of collecting the IP addresses of incoming SSH connections, username-password pairs (for authentication), and logs of every command along with the generated responses by the model. By incorporating a LLM, our custom SSH server can engage attackers in a realistic manner, providing insights into their actions and enhancing the honeypot's overall functionality.

C. Training Loss Analysis

As illustrated in Figure 4, the training losses of our fine-tuned model exhibit a steady decline over the training steps. This trend indicates that the model effectively learned from our dataset and adapted well to the task of mimicking a Linux server. During the fine-tuning phase, we employed a learning rate of 5×10^{-4} and conducted a total of 36 training steps. The entire training process was completed in 14 minutes. The consistent decrease in training loss demonstrates the model's capability to improve its performance progressively, thereby enhancing its ability to generate realistic and contextually appropriate responses.



```
○ (base) ai-lab@ailab:~$ ssh -T -p 2222 "root@localhost"
  root@localhost's password:
  root@localhost:~/$ ls -al

  total 32
  drwxr-xr-x  3 root root 4096 Apr  2 14:30.
  drwxr-xr-x 14 root root 4096 Apr  2 14:30..
  -rw-r--r--  1 root root  220 Apr  2 14:30.bashrc
  -rw-r--r--  1 root root  807 Apr  2 14:30.profile
  -rw-r--r--  1 root root  220 Apr  2 14:30.bash_logout

  root@localhost:~/$ echo 'hello world'
  hello world
  root@localhost:~/$ ifconfig
  eth0      Link encap:Ethernet  HWaddr 00:11:22:33:44:55
            inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

  lo        Link encap:Local Loopback
            inet addr:127.0.0.1  Mask:255.0.0.0
            UP LOOPBACK RUNNING  MTU:16436  Metric:1
  root@localhost:~/$ █
```

Fig. 3. Example of Honeypot SSH Connection

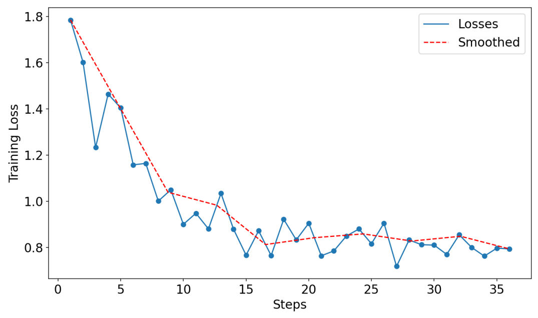


Fig. 4. Training losses over 36 steps in Supervised Fine-Tuning

D. Similarity Analysis with Cowrie Outputs

To evaluate the performance of our fine-tuned language model, Llama3-8B, we employed multiple metrics to measure the similarity between the expected (Cowrie) and generated terminal outputs. We used cosine similarity to quantify the cosine of the angle between two vectors in high-dimensional space, with higher scores indicating better performance. Additionally, we used the Jaro-Winkler similarity, which measures the similarity based on matching characters and necessary transpositions, with higher scores indicating closer matches. Finally, we utilized the Levenshtein distance, which calculates the minimum number of single-character edits needed to transform one string into another, with lower scores indicating closer matches. These diverse metrics provided a comprehensive evaluation of our model's performance.

The results of our evaluation, summarized in Table I, demonstrate the performance of our fine-tuned language model, Llama3-8B, using different similarity and distance metrics over 140 random samples. The fine-tuned model achieved a cosine similarity score of 0.695 (higher is better), indicating a strong match between the expected and generated terminal outputs. The Jaro-Winkler similarity score was 0.599 (higher is better), also reflecting a reasonable level of similarity. The Levenshtein

TABLE I
SIMILARITY AND DISTANCE METRICS

Metric	Mean Score	
	Base	Fine-Tuned
Cosine Similarity	0.663	**0.695**
Jaro-Winkler Similarity	0.534	**0.599**
Levenshtein Distance	0.332	**0.285**

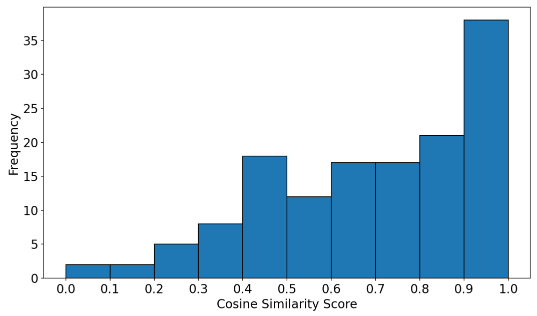


Fig. 5. Histogram of Cosine Similarity Scores over 140 Samples

distance was 0.332 (lower is better), suggesting a relatively low number of edits needed to align the generated output with the expected one.

In addition, fine-tuned LLM showed improvements across all metrics compared to the base model. These results show the effectiveness of our model in generating outputs that closely match the expected responses from a Cowrie honeypot server.

As shown in Figure 5, the cosine similarity scores of the outputs generated by LLM exhibit a distribution with most scores concentrated towards higher values, indicating that the model's responses are mostly similar to the expected outputs.

Some outputs may vary but remain contextually accurate and true to the commands. The LLM honeypot server consistently handles out-of-bound or invalid commands due to strict system prompt guardrails and training on erroneous examples. For instance, when encountering unrecognized commands, the model replicates realistic system behavior with messages like 'bash: XXX: command not found.' Testing shows the model maintains its persona and provides convincing responses, even with unexpected inputs.

IV. Conclusion

This study introduces an innovative approach to developing interactive and realistic honeypot systems using Large Language Models (LLMs). By fine-tuning a pre-trained open-source language model on attacker-generated commands and responses, we created a sophisticated honeypot that enhances realism and deployment effectiveness.

Our LLM-based honeypot system improves response quality and the ability to detect and analyze malicious activities. The integration of LLMs with honeypot technology creates a dynamic, adaptive system that evolves with emerging threats. Leveraging LLMs' reinforcement learning and attention mechanisms, our system refines responses and maintains high contextual relevance, providing deeper insights into attacker behavior and strengthening security infrastructures.

References

[1] N. Naik, C. Shang, P. Jenkins, and Q. Shen, "D-fri-honeypot: a secure sting operation for hacking the hackers using dynamic fuzzy rule interpolation," *IEEE Transactions on Emerging Topics in Computational Intelligence*.
[2] M. A. Hakim, H. Aksu, A. S. Uluagac, and K. Akkaya, "U-pot: a honeypot framework for upnp-based iot devices," *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*, 2018.
[3] M. L. Bringer, C. Chelmecki, and H. Fujinoki, "A survey: recent advances and future trends in honeypot research," *Int. Jrnl of Comp. Network and Inf. Security*, 2012.
[4] C. Kelly, N. Pitropakis, A. Mylonas, S. McKeown, and W. J. Buchanan, "A comparative analysis of honeypots on different cloud platforms," *Sensors*, 2021.
[5] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Schöll, and N. P. Smart, "Computer security – esorics 2021," *Lecture Notes in Computer Science*, 2021.
[6] A. R.A., D. F.M., A. B.K., A. O.S, and O. T.J, "Improving deception capability in honeynet through data manipulation," *Internet Technology and Secured Transaction*, 2015.
[7] H. Griffioen, K. Oosthoek, P. v. d. Knaap, and C. Doerr, "Scan, test, execute: adversarial tactics in amplification ddos attacks," *The 2021 ACM SIGSAC Conference on Computer and Communications Security*.
[8] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, S. Zhong, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *ACM Transactions on Knowledge Discovery from Data*, 2024.
[9] F. McKee and D. Noever, "Chatbots in a Honeypot World."
[10] U. Sedlar, M. Kren, L. Štefanič Južnič, and M. Volk, "CyberLab honeynet dataset," 2020.
[11] M. Sladić, V. Valeros, C. Catania, and S. Garcia, "LLM in the Shell: Generative Honeypots," 2023.
[12] M. Oosterhof, "Cowrie 2.5.0 documentation." [Online]. Available: https://cowrie.readthedocs.io/
[13] C. Kelly, N. Pitropakis, A. Mylonas, S. McKeown, and W. J. Buchanan, "A comparative analysis of honeypots on different cloud platforms," *Sensors*, 2021.
[14] S. Deshmukh, R. Rade, and D. F. Kazi, "Attacker behaviour profiling using stochastic ensemble of hidden markov models," 2019.
[15] V. Švábenský, J. Vykopal, P. Seda, and P. Čeleda, "Dataset of shell commands used by participants of hands-on cybersecurity training," *Data in Brief*, 2021.
[16] "Linux man pages tldr summarized." [Online]. Available: https://huggingface.co/datasets/tmskss/linux-man-pages-tldr-summarized/
[17] H. Touvron, L. Martin *et al.*, "Llama 2: Open foundation and fine-tuned chat models."
[18] Microsoft, "Phi-3 technical report: A highly capable language model locally on your phone," 2024.
[19] MetaAI, "Code llama: Open f. models for code," 2024.
[20] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, and Others, "Mixtral of experts," 2024.
[21] Y. Zheng, R. Zhang, J. Zhang, Y. Ye, Z. Luo, and Y. Ma, "Llamafactory: Unified efficient fine-tuning of 100+ language models," 2024.
[22] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021.
[23] N. Jain, P. yeh Chiang *et al.*, "Neftune: Noisy embeddings improve instruction finetuning," 2023.
[24] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," 2023.
[25] "Paramiko documentation." [Online]. Available: https://www.paramiko.org/
[26] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, and Others, "Huggingface's transformers: State-of-the-art natural language processing," 2020.