IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 29 August 2024, accepted 24 September 2024, date of publication 2 October 2024, date of current version 14 October 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3472460

APPLIED RESEARCH

Don't Stop Believin': A Unified Evaluation Approach for LLM Honeypots

SIMON B. WEBER[1], **MARC FEGER**[1], **AND MICHAEL PILGERMANN**[2]
[1]Department of Computer Science, Heinrich-Heine-Universität Düsseldorf, 40225 Düsseldorf, Germany
[2]Department of Computer Science and Media, Brandenburg University of Applied Sciences, 14770 Brandenburg an der Havel, Germany
Corresponding author: Simon B. Weber (Simon.Weber@hhu.de)

ABSTRACT The research area of honeypots is gaining new momentum, driven by advancements in large language models (LLMs). The chat-based applications of generative pretrained transformer (GPT) models seem ideal for the use as honeypot backends, especially in request-response protocols like Secure Shell (SSH). By leveraging LLMs, many challenges associated with traditional honeypots – such as high development costs, ease of exposure, and breakout risks – appear to be solved. While early studies have primarily focused on the potential of these models, our research investigates the current limitations of GPT-3.5 by analyzing three datasets of varying complexity. We conducted an expert annotation of over 1,400 request-response pairs, encompassing 230 different base commands. Our findings reveal that while GPT-3.5 struggles to maintain context, incorporating session context into response generation improves the quality of SSH responses. Additionally, we explored whether distinguishing between convincing and non-convincing responses is a metrics issue. We propose a paraphrase-mining approach to address this challenge, which achieved a macro F1 score of 77.85% using cosine distance in our evaluation. This method has the potential to reduce annotation efforts, converge LLM-based honeypot performance evaluation, and facilitate comparisons between new and previous approaches in future research.

INDEX TERMS IT security, honeypot, large language model, GPT, cosine distance, evaluation.

I. INTRODUCTION

Honeypots are widely used in research to analyze attacker behavior and in industry contexts to detect and prevent attacks. For several protocols, medium or high-interaction honeypots are prevalent and usable. However, developing high-interaction honeypots is a tedious task, especially if the honeypot should mimic a specific device or service. In addition, attackers often specialize in honeypot detection and develop tests to identify whether a listening service is a honeypot [1]. As a result, using publicly available or well-known honeypots can alienate sophisticated attackers.

Large Language Models (LLMs) have recently gained recognition as valuable tools across various domains, with IT security being a newly emerging area of application [2], [3], [4]. Evaluations have shown that ChatGPT (GPT-3.5 and GPT-4) are capable of contextualizing conversations, producing source code, and generating other machine output [5], [6], [7]. Despite the diversity of data sources, GPT-3's training dataset includes common web data, among others, which encompass technical manuals, forums, and coding resources [5], [8], suggesting their applicability in honeypot systems.

Since server attacks often resemble technical conversations, with attackers sending commands and expecting specific responses, LLMs might be well-suited to mimic these exchanges and enhance honeypot interaction. With their ability to understand context and generate a wide variety of applicable responses, they could be remarkably effective when an attacker may input unexpected or uncommon commands.

In particular, protocols with a request-response format, such as SSH, appear to be predestined for chat-based LLMs. Compared to conventional high-interaction honeypots for these protocols, using LLMs for honeypots could have the advantage of reducing the risk of unintentional breakouts.

The associate editor coordinating the review of this manuscript and approving it for publication was Diana Gratiela Berbecaru.

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

Traditional honeypots, designed to replicate real systems, operate within an actual OS environment, which carries the inherent risk that an attacker, upon discovering the honeypot, could exploit vulnerabilities in the underlying OS to escape its confines. Such a breakout would compromise not only the honeypot but also pose a serious threat to the broader network infrastructure. In contrast, LLMs are no real OS; they generate responses based on training data without executing commands in a real system environment. Therefore, an attacker interacting with an LLM-based honeypot can only engage with the simulated responses generated by the model. Consequently, this approach offers a high level of interaction and realism and ensures a safer and more secure deployment.

The potential for using LLM-based honeypots is even greater when applied to sector- or application-specific protocols, where developing high-interaction honeypots is particularly challenging due to the limited availability of preliminary work. While [2] demonstrated the basic potential of LLMs in simulating various OS environments (Windows, Linux, Mac) and applications (e.g., Jupyter notebooks, TeamViewer), subsequent studies have shown that LLMs can also imitate more specialized environments: MySQL servers [9], IoT devices [10], and even sector-specific devices using protocols such as Modbus and S7comm [11].

Recent research indicates that while code generated by LLMs may appear correct at first glance, it is often prone to errors [12]. Previous studies on LLM-based honeypots have shown that this is also true for this research area, and not all responses generated by LLMs are accurate, suggesting that they are not yet reliable enough to serve as flawless high-interaction honeypots [3], [13].

We believe, in the context of honeypots, convincing the attacker takes precedence over achieving flawless accuracy. In realistic scenarios, though, it is essential that the attacker remains unaware of the honeypot's nature for as long as possible. This underscores the importance of evaluating how effectively LLMs can currently function as honeypots and identifying the areas where they still face challenges.

In fact, all prior research approaches to performance evaluation of LLM-based Honeypots are different. Some rely on expert analysis, having humans assess the performance of LLMs [2], [3]. Others create unique metrics, comparing LLM outputs to real server responses at the character or byte level [9], [11], or by contrasting LLM-based honeypots with traditional ones like Cowrie.[1]

In the latter case, comparison methods differ, using metrics like successful sessions, average session length, or response success rates [4]. Another approach involves using real server responses as a baseline, comparing the Levenshtein distance and L-ratio between medium-interaction honeypot responses and those generated by LLMs to evaluate performance [13].

Despite their pioneering work, current research on LLM-based honeypots mostly focuses on individual responses without considering the context or the ability to mimic a real system. Although the initial evaluations are valuable, they do not allow for comparisons between different approaches, nor do they clarify whether LLM improvements enhance honeypot effectiveness. As a result, the absence of standardized evaluation metrics limits the ability to assess and validate the true impact of these advancements on honeypots.

At present, there are no established evaluation baselines or ground truth data available for assessing Secure Shell (SSH) sessions, especially those that involve real-world attacks.

With this in mind, we focus on evaluating the performance of GPT-3.5 in mimicking SSH servers as components of honeypots in IT security. Rather than demonstrating its ability to function as a genuine system, our goal is to evaluate how to measure the performance of this fundamental yet state-of-the-art LLM, which serves as the foundation for many rapidly evolving derivative models and represents the first to explore.

To do this, we investigate how well GPT-3.5 can replicate an SSH server, particularly in providing responses that align with attacker requests, maintaining context across multiple interactions, and identifying where the model succeeds or fails. We then explore methods to differentiate between accurate and flawed responses.

In our work, we combine and extend three existing datasets to (1) ensure comparability with prior approaches [3], (2) identify GPT-3.5's limitations with complex single-line commands [14], and (3) maintain realism by including real-world attacks and server responses [15].

As part of advancing the global evaluation of GPT-based honeypots, particularly regarding context-dependent interactions, we contribute:

- An annotation framework[2] for evaluating GPT-based responses as SSH servers, including those involving multiple interactions requiring context, grounded in the Cambridge Dictionary's definition of what is convincing.[3]
- Validation of our framework on a subsample of 7,000+ request-response pairs, resulting in 1,400+ unique annotations covering 230 base commands and interaction chains from three state-of-the-art honeypot datasets. GPT-3.5 responses were annotated by five experts, achieving an average Krippendorff's α of 57.4.
- An investigation into GPT-3.5's difficulty in maintaining context over multiple attacker interactions, which can make a honeypot detectable.
- Suggesting that annotation effort and GPT-3.5's performance might be improved through a paraphrase-mining approach, potentially allowing for the distinction between genuine and impostor responses with 77.85% macro F1 using cosine distance.

II. DATA RESOURCES

Our first dataset, sourced from the work of [3], involved human participants interacting with an LLM-honeypot

[1] github.com/cowrie/cowrie

[2] github.com/TomatenMarc/Dont-Stop-Believin.git

[3] dictionary.cambridge.org/dictionary/english/convincing

system, resulting in 226 unique commands, and is referred to as the Prague dataset, named after TU Prague. Participants used SSH for tasks like package management, file system, and network operations to evaluate their ability to distinguish shelLM outputs from those of a real system.

The second dataset, NL2Bash by [14], was constructed to facilitate the translation of natural language (NL) sentences into Bash commands. It comprises 12,609 text-command pairs initially collected from various web sources such as forums, tutorials, and tech blogs. The dataset includes Bash commands utilizing over 100 unique utilities and corresponding high-quality English descriptions provided by expert Bash programmers. It was used to train and evaluate various neural semantic parsing models, demonstrating the complexity and challenges of mapping NL to Bash commands.

The third dataset, collected to analyze attacker behaviors on a high-interaction Linux SSH honeypot [15], is referred to as the Halle dataset, named after the researchers from Halle University. It was collected over two distinct periods, from May 2017 to September 2019 and January to October 2021. It encompasses the subsequent commands executed by attackers and the responses from an SSH honeypot.

The original datasets can be found or requested from the authors of [3], [14] and [15] are chosen because:

1) **Prague** [3] contains commands used in prior research regarding LLM-based honeypots to have a baseline for comparison.
2) **NL2Bash** [14] includes complex and challenging single-line commands. Responses to those exact commands are usually not directly publicly accessible. The LLM must, therefore, logically capture the meaning of the command and estimate the response.
3) **Halle** [15], captured from real-world attackers interacting with high-interactive SSH honeypots, is the most realistic dataset of these three, containing several coherent commands in a sequence (SSH sessions). This enables us to investigate the influences of contextualization towards the LLM's performance.

For the two datasets without valid server responses (Prague and NL2Bash), we sent all commands to a virtual machine and collected the responses (stdout and stderr). To ensure comparability, we used the same OS (Debian Jessie) as the researchers that created the Halle dataset [15] because responses to the same commands can vary between different Linux distributions. For instance, the `ps` command on Debian might show a different format or include different default columns compared to Fedora or Arch Linux due to differences in how system tools and libraries are configured.

In preparation of the Halle data, we filtered out those commands and responses that contained non-printable characters, e.g., if the attacker used the SFTP subsystem. We then removed duplicates; for example, if an attacker first used the command `id` and then checked system or kernel information via `uname` and then interrupted the connection, we removed all sessions except one. For NL2Bash and Prague, we excluded all commands that produced empty responses from the virtual machine, as these do not meet the criteria for evaluating the usability of metrics via representations. In contrast, empty responses were retained for Halle, as they could be important for the session state, such as when changing the directory directly impacts subsequent commands and responses. Especially for this dataset, every command from each session was provided to the LLM, along with the context of previous commands and responses, ensuring that the LLM could account for the server's state throughout the session.

A. SAMPLE GENERATION

We created a representative sample for each dataset, preserving the original distribution of command length, base commands, and labels. This approach maintained nearly identical mean command lengths and session sizes (for the Halle dataset) while including at least one entry for every base command with a non-empty response. To identify the base commands within the data, we used the method described in [14]. We stripped `sudo` from the beginning of a command and replaced absolute path names with their base names (e.g., `/bin/find` to `find`).

We included commands that are not valid on a command line, such as those in NL2Bash, that remained in plain text rather than being processed into shell commands, assuming a real server to throw errors in such cases. In contrast, the LLM might provide a server response matching the intention [3].

Table 1 provides the differences between the original datasets and the generated sample.

TABLE 1. Summary of dataset and sample distributions. Note that the total count of base commands is not the sum of unique base commands across all datasets, as some base commands appear in multiple datasets.

Dataset	Source	Commands	Unique Base-Commands
Prague	Original	226	67
Prague	Sample	129	49
NL2Bash	Original	12,607	226
NL2Bash	Sample	334	183
Halle	Original	2,483	85
Halle	Sample	943	59
Total	Original	15,316	251
Total	**Sample**	**1,406**	**230**

B. LLM RESPONSE GENERATION

We employed *GPT-3.5 Turbo*[4] to simulate an SSH server operating on a Debian Jessie server, chosen specifically because all three datasets utilized this OS, ensuring consistency across our annotations and experiments.

The prompt was meticulously designed to preclude any supplementary commentary or elaboration, thereby guaranteeing that the outputs produced by the language model remained indistinguishable from those generated by an authentic SSH server. To further enhance the realism of the simulation, the prompt explicitly directed the language

[4]platform.openai.com/docs/models/gpt-3-5-turbo

model to generate contextually appropriate error messages in response to incorrect or malformed commands, effectively emulating the behavior of a genuine server environment.

To leverage the advantages of few-shot learning [8], we provided an example interaction to guide the LLM in understanding the expected format and behavior. This example, combined with a carefully structured prompt, was designed to ensure consistency and realism in the LLM's responses throughout the session. The complete prompt, along with the code to retrieve responses from GPT-3.5, our detailed evaluation, sample datasets, and annotation materials, can be accessed in our repository[2].

III. ANNOTATION FRAMEWORK

As a baseline, five annotators rated the generated responses from the LLM. To ensure consistency in the evaluation process, an annotation guide containing rating instructions and examples of edge cases, carefully deliberated and tested by the main authors, was provided. The annotators were trained students with at least a bachelor's degree in computer science.

The annotators evaluated whether the LLM-generated responses were convincing when compared to real responses from an SSH server. In this context, and according to the Cambridge Dictionary[3], convincing means that the LLM is *able to make you believe that something is true or right*, thereby appearing as if it is a real SSH server.

It is important to note that this does not necessarily equate to technical correctness. A response can be considered convincing even if it contains technical inaccuracies as long as it aligns with the general expectations of how an SSH server might respond under similar circumstances. The focus is on the likelihood of deceiving a potential attacker into thinking they are interacting with a real SSH server.

For this task, a binary annotation system was employed. Annotators rated each response as either convincing or not. Each data point that required annotation consisted of a three-part format designed to simulate an SSH session interaction. This format provided the context for evaluating the LLM's ability to mimic real SSH server responses accurately:

- **Completion Request**: This represents an SSH session history, providing context for the LLM and the annotators. It follows the user-assistant dialogue format, with *user:* <*ssh-command*> and *assistant:* <*server-response*>. The session history concludes with a command from a user to which the LLM is supposed to generate a response.
- **Expected Response**: This is the output of the real SSH server in response to the SSH command sequence. It served as a reference for what a typical SSH server might return but was not shown to the LLM. This expected response helped the annotators gauge the accuracy and realism of the LLM's generated response.
- **Completion Response**: This is the LLM-generated response based on the completion request. It is the focus of evaluation. Although the expected response serves as a benchmark, any plausible response that differs from the expected response was still considered valid.

For further details on the annotation, the associated materials, or the results, please refer to our repository[2].

IV. FINDINGS

A. ANNOTATION EVALUATION

In total, the five annotators evaluated 7,030 request-response pairs (1,406 each). Slightly more than half of the pairs were deemed convincing by the annotators, resulting in a fair label distribution. An examination of the individual datasets reveals some differences, as Table 2 shows. The best results were yielded by the Prague dataset with 64% convincing, while the NL2Bash dataset yielded the worst with just 40%.

TABLE 2. Results of the annotation the LLM responses as convincing.

Dataset	Convincing (%)
Prague	64.34
Halle	54.93
NL2Bash	40.12
Overall	52.28

The agreement among the annotators, measured using Krippendorff's strict α, resulted in an average agreement of 57.4 α, which we consider solid given the task's difficulty and the complexity of the data. While the pairs associated with the Prague and NL2Bash datasets showed lower agreement, the Halle dataset achieved a higher agreement of 63.46α, as detailed in Table 3.

TABLE 3. Comparison of Inter-Annotator Agreements across the datasets, measured using Krippendorff's α. The overall agreement is calculated as the average performance across the individual datasets.

	Krippendorff's Alpha (%)
Halle	63.46
NL2Bash	43.90
Prague	40.71
Overall	57.38

B. DATASET COMPARISON

By analyzing the statistical characteristics of the three datasets, we aimed to understand the on-par label distribution. Given that all general parameters, including the prompt and LLM model, were consistent across the three datasets, with the only variations being the commands sent to the LLM and the session history in the Halle dataset, our analysis centered on these specific factors.

First, we analyzed whether certain base commands resulted in a higher level of convincingness than others. The most convincing commands across at least two datasets with over

20 instances were `whoami`, `crontab`, and `uname`, each rated as convincing in over 90% of cases.

On the other end, we observed base commands mostly performing as non-convincing, such as `w` (4% convincing), `wget` (8% convincing), and `top` (17% convincing).

For `wget` and `top`, the reasons are quite obvious. `wget` is used to download a file from a remote location to the server. The LLM mostly hallucinated the download of that file or stated that the file could not be downloaded for various reasons: not found, DNS not available, etc.

The `top` command retrieves current processes in an interactive environment. The corresponding man page[5] describes that the program provides a dynamic real-time view and requires the use of cursor control keys, such as the arrow keys. Such interactive elements cannot be realized in the current chat-based implementation of GPT-3.5, as it lacks the ability to continuously process and visually display real-time inputs and outputs, as would be needed in an interactive environment. For this reason, attempts to simulate such functions result in error messages. These errors were often rated unbelievable by the annotators (e.g., `top: error while loading shared libraries`).

Similar issues arise with the `w` command, which is meant to display system uptime and information about active users. The LLM had difficulties with this command and mostly stated that it was not found or returned nothing. As it usually returns up-time and logged-in users, including the attacker's connection, the annotators considered this response unconvincing. In contrast to the other unconvincing commands, the reasons here are not obvious. The command is prevalent, non-interactive, and reference responses can be found in several places. One reason could be that the one-letter command is just too short to be unambiguous.

C. COMMAND COMPLEXITY

Driven by the analysis of individual commands, we now focus on their complexity, as our data indicated that certain commands exhibited varying levels of persuasiveness across different datasets. This variation is particularly evident with the common command `cd`, which was rated as 100% convincing in the Prague dataset, 57% in the Halle dataset, and not convincing at all in the NL2Bash dataset.

To quantify this complexity, we considered the command length and the frequency of certain characters that have a special meaning in a command line and SSH environment [16]. We assume longer commands signal complex operations more often. By using special characters, commands can perform multiple tasks in a single line, handle complex logic, and efficiently manipulate data streams, enhancing both their functionality and perceived complexity.

The mean command length varied across the datasets, with Halle having the longest commands at 285.6 characters, followed by NL2Bash at 47.2 characters, and Prague with the shortest commands at 13.4 characters. Focusing on the median reveals that Halle has some large outliers, but the median command length (37 characters) is close to that of NL2Bash (43 characters). Prague's median command length is just slightly below the mean, with 12 characters. However, all command lengths are skewed to the left. When comparing command lengths across datasets, one can observe that Halle and NL2Bash have similar lengths, whereas the Prague dataset primarily consists of shorter commands (Figure 1).

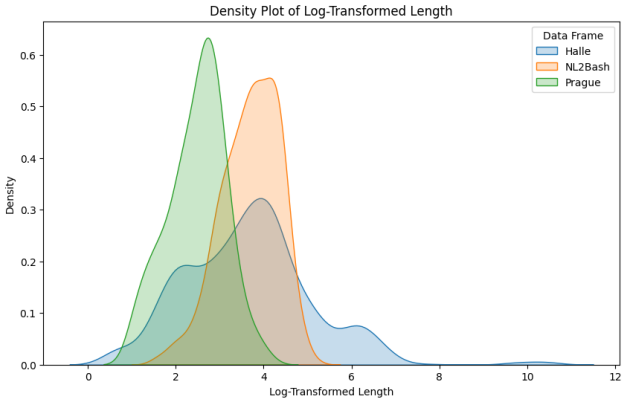

FIGURE 1. Density plot of log-transformed command length distribution.

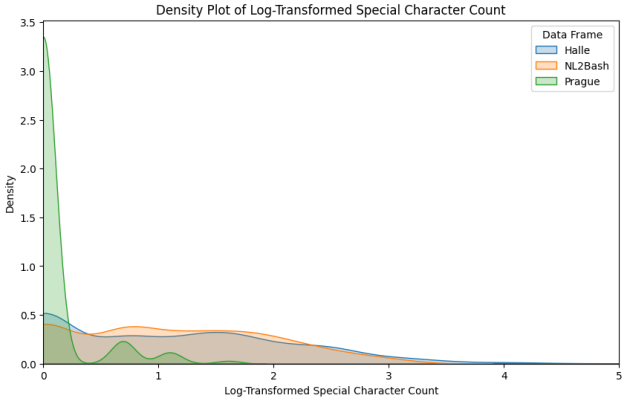

FIGURE 2. Density plot of log-transformed special character frequency.

Command length seemingly affects convincingness. Commands longer than the median length exhibit reduced convincingness: 39% for Halle, 32% for NL2Bash. The Prague dataset also shows a slight decrease in convincingness for commands above median length but maintains 59% convincingness. This slightly lower drop in convincingness might be due to the maximum command length of just 56 characters.

The presence of certain characters noticeably reduces the convincingness. Commands containing at least one of the characters ';', '$', '|', '&', or '<' are only 40% convincing on average in the combined dataset. The presence of the substitution character '$' reduces the convincingness to 37%, parentheses to 29%, and square brackets to 26%.

[5]https://manpages.org/top

Convincingness declines as the number of special characters increases: commands with more than three pipes are convincing in 22% of cases, more than three parentheses in 19%, four or more semicolons or braces in 14%, and four or more angle brackets in 11%.

While the overall presence of special characters generally indicates lower convincingness, no specific group of characters consistently harms the LLM's performance. For instance, in the NL2Bash dataset, the presence of three or more substitution characters '$' results in 8% convincingness, whereas in the Halle dataset, 50% of commands with the same count of substitution characters remain convincing.

We also analyzed the correlation between the use of special characters and convincingness using Kendall's τ. The relationship is weak but statistically significant, with a correlation of -0.21 (p = 1.509e-19) in the overall sample data. Similarly, the correlation between command length and convincingness is also weak but statistically significant, with a correlation of -0.22 (p = 1.007e-22).

Comparing the three datasets, commands in Halle and NL2Bash are more complex than in Prague. The mean number of special characters per command is 6.4 for Halle, 1.5 for NL2Bash, and 0.1 for Prague. Despite some commands in Halle having many special characters, the median in both Halle and NL2Bash is just one, indicating most commands are similar in complexity.

Thus, Halle and NL2Bash are similarly complex in terms of command length and special character count, while the Prague commands are shorter and contain fewer special characters. The distribution of the special characters counts in commands through the datasets can be seen in Figure 2.

D. SESSION STATE TRANSITIONS

To further investigate the differences in convincingness, we examined the role of the session state on the convincingness of the generated responses.

Halle stands out as the most realistic dataset, sourced from real attacker sessions, and is unique in being the only session-based dataset with commands embedded within a session history. For this reason, we focus on Halle, examining transition probabilities to gain deeper insights into its varying scores, specifically analyzing how preceding commands influence the convincingness of subsequent commands within a session.

The data reveals that when a response is not convincing, the likelihood of the subsequent response being believable is only marginally higher (50.96%). However, when a response is convincing, the probability that the following response will also be convincing rises to 59.59%. These findings are detailed in Table 4. The results suggest that while the LLM shows some improvement in generating convincing responses with additional context, there is still a significant risk of inconsistency. This underscores the importance of developing methods to reliably distinguish between genuine and impostor responses, as relying solely on context may not be sufficient to ensure consistently convincing outputs.

TABLE 4. Transition probabilities of session state for Halle.

$p(s+1\|s)$	Non-Convincing	Convincing
Non-Convincing	49.04%	50.96%
Convincing	40.41%	59.59%

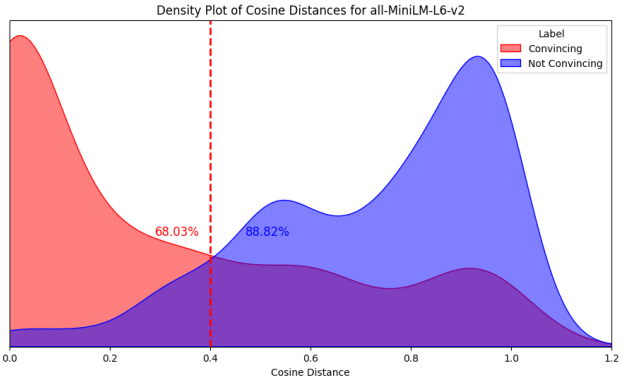

FIGURE 3. Density plot of cosine distance illustrating the separation between convincing and non-convincing responses via all-MiniLM-L6-v2.

E. DISTANCE AND REPRESENTATIONS

To enhance future evaluation of LLM-based honeypots, we conducted a series of ablation experiments with various representation models to establish best practices. By examining the structural differences and similarities between labels, specifically what defines a convincing response versus a non-convincing one in the given context, we aim to gain valuable insights into the measurability of these distinctions.

Sentence-BERT (SBERT) [17] can be used alongside different pre-trained transformer models for comparing text representations. SBERT extends the BERT architecture [18] by utilizing Siamese network structures [19] to produce semantically meaningful sentence embeddings. When these embeddings are combined with cosine distance, a standard method for comparing text representations [20], the classification problem becomes one of metric learning [21], focusing on the measurability of similarities. The classifier leverages pre-trained representations, calculates their distances to their baseline counterparts, and identifies an optimal threshold for effective class separation, thereby streamlining the training.

Cosine distance, $d(v_1, v_2) = 1 - \cos(v_1, v_2) \in [0, 2]$, measures the semantic similarity between two vectors v_1 and v_2 by calculating the cosine of the angle between them ($\cos(v_1, v_2) \in [-1, 1]$) and mapping the result to a range from 0 (similar) to 2 (dissimilar) for better interpretability. When applied to SBERT-generated vector representations, this distance quantifies the semantic similarity between LLM-generated responses (v_1) and baseline outputs (v_2).

We tested various models for classifying the convincingness of LLM outputs compared to the baseline virtual machine outputs using SBERT and cosine similarity.

The evaluation relied on macro F1 scores to assess the models, giving equal importance to both classes, convincing and unconvincing responses. This method ensures that the analysis fairly considers the model's performance across the entire binary classification task without favoring one class over the other.

The models evaluated include all-MiniLM-L6-v2,[6] bert-base-uncased,[7] distilbert-base-uncased,[8] and roberta-base.[9] The macro F1 values for each model are as follows:

- **all-MiniLM-L6-v2:** 77.85%
- **bert-base-uncased:** 75.36%
- **distilbert-base-uncased:** 75.15%
- **roberta-base:** 67.96%

The all-MiniLM-L6-v2 model achieves the highest macro F1 score of 77.85%, outperforming the remaining models with bert-base-uncased and distilbert-base-uncased slightly lower performances. An optimal cosine distance threshold of 0.4 for all-MiniLM-L6-v2 shows that 68.03% of convincing responses have lower distances, while 88.82% of unconvincing ones have higher distances, indicating better semantic similarity for convincing responses, as evidenced in Table 3. All details on all evaluation steps can be viewed in our repository[2].

V. DISCUSSION

We observed differences in the annotation results of the Prague dataset compared to previous reports. Since [3] used a slightly different analysis approach, we first had to align their results with ours. Thereby, we define convincing as the sum of the true-negative (74%) and false-negative (0.5%) rates, considering any response the Prague annotators deemed appropriate as convincing (even if their experts stated it was revealing the honeypot). However, our experts found only 64.3% of the Prague sample to be convincing, compared to the 74.5% in the previous analysis.

Differences in annotation results are common, but the dataset's pre-filtering may have also influenced these outcomes. Removing empty responses might have excluded simpler commands, like creating new folders, changing directories, or deleting files, decreasing perceived convincingness.

To our knowledge, this study includes the largest number of unique base commands used to evaluate such a system, covering the 100 essential command-line utilities.[10] This categorization showed that some base commands yielded higher convincingness while others did not, which we consider a natural phenomenon rather than a flaw. This is consistent with the findings from other studies that have used base commands [3] or command groups [13] for analysis.

However, discrepancies in convincingness between the datasets were observed in the base-command categorization. Listing 1 illustrates the issue.[11]

```
cat /proc/cpuinfo \
| grep name \
| head -n 1 \
| awk '{print $4,$5,$6,$7,$8,$9;}'
```

LISTING 1. A sample command from the Halle dataset.

Although the base command is `cat`, the LLM must also process three other shell commands. Errors from these could mistakenly appear as if they originated from the base command `cat`. Therefore, the base command categorization better describes dataset variation rather than defining the LLM's limitations as a honeypot.

We examined command complexity, finding that the presence of special characters and longer commands correlate with reduced convincingness. The Prague dataset's lower length and complexity may explain its higher scores. However, this does not account for why similar datasets like Halle and NL2Bash do not achieve the same results. The session history might explain this, as transition probabilities indicate that once the LLM generates a convincing response, the likelihood of subsequent convincing responses increases.

Convincingness and annotator agreement fluctuated with more complex datasets. This may be because annotators, like the LLM, struggled with command complexity, and session context aided both. This might explain the higher scores and agreement compared to the NL2Bash and Prague datasets.

However, using SBERT and cosine distance to evaluate GPT-3.5's responses proves effective for future LLM assessments, potentially reducing the need for human annotation.

VI. CONCLUSION AND FUTURE WORK

In this study, we explored the frontiers of LLM-based honeypots, presenting a novel approach for the performance analysis of such systems. Our findings reveal substantial variability in the performance of GPT-3.5 when employed as a honeypot backend, with convincingness rates across the three datasets ranging from 40% to 64%.

This variability was linked to command complexity; both between and within datasets, longer commands or those with more special characters consistently led to less convincing responses. We highlight GPT-3.5's clear limitations in handling complex SSH commands, rendering it unsuitable as a high-interaction honeypot backend in its current state.

Nevertheless, our study also demonstrates that leveraging the context window of this LLM can significantly enhance overall convincingness, particularly in session-based protocols like SSH, where an initial believable response increases the likelihood of subsequent convincing responses.

[6] huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[7] huggingface.co/google-bert/bert-base-uncased
[8] huggingface.co/distilbert/distilbert-base-uncased
[9] huggingface.co/FacebookAI/roberta-base
[10] oliverelliott.org/article/computing/ref_unix/
[11] Line breaks inserted for improved readability.

Our investigation explored whether assessing the convincingness of generated responses is fundamentally a metric issue. We introduced a new approach using the distance between generated and genuine responses as a measure of believability, with paraphrase mining achieving a macro F1 score of 77.85%. This method shows promise for reducing manual annotation, streamlining LLM performance evaluation, and facilitating comparisons with previous honeypot research. Additionally, it could serve as a pre-filtering tool, automatically rejecting unlikely responses and allowing re-queries, potentially improving overall performance.

Taken together, our work sets the stage for future research comparing LLMs as honeypots, particularly exploring whether fine-tuned models can better handle complex command-line instructions. This is especially relevant for specialized domains like healthcare, where improved interaction fidelity with protocols like HL7 could be highly beneficial. Additionally, our findings suggest further exploration into how these enhancements can keep attackers engaged longer without detecting the honeypot, advancing the effectiveness of LLM-based IT security.

VII. LIMITATIONS

We identified key limitations in the LLM's practical deployment, particularly with handling time-dependent responses and latency issues. Several commands require the inclusion of real-time data, such as the current date or time, which the LLM lacks awareness of. This gap necessitates integration with external systems or APIs to provide accurate, real-time information. Additionally, while latency was negligible in our theoretical setup, it becomes a significant risk in real-world applications. If the LLM takes too long to respond – especially during complex operations – this delay could tip off an attacker that something unusual is happening. In our study, response generation occasionally took tens of seconds, a delay that would be unacceptable in a real-world scenario where speed is critical. Both issues warrant further investigation, as they are not only relevant for honeypots but also represent broader challenges in LLM research (e.g., [22], [23]).

GPT-3.5's limited context window restricts session history, so we capped Halle dataset sessions at 50 command-response pairs. Longer sessions could be handled through methods like Rotary Position Embeddings and prompt compression [24]. Newer OpenAI models with larger context windows may mitigate this issue, and future studies could benefit from these improved models.

Furthermore, interactive environments challenge LLMs due to limited chat-based interactivity. Solutions include deactivating interactive elements or creating a wrapper to simulate interactions, such as with the vi editor or top.

As research assistants, the annotators carried out the annotation during working hours and were paid appropriately for their time. AI text generation tools, including ChatGPT, were utilized solely for experimental purposes and did not contribute to the writing of this article.

ACKNOWLEDGMENT

(Simon B. Weber and Marc Feger contributed equally to this work.) The authors would like to thank their annotators for their expert contributions and the anonymous reviewers for their fair and insightful assessments, which have greatly improved their work. Your willingness to share your expertise and provide constructive feedback is deeply appreciated.

REFERENCES

[1] M. Abbas-Escribano and H. Debar, "An improved honeypot model for attack detection and analysis," in *Proc. 18th Int. Conf. Availability, Rel. Secur.*, vol. 201, Aug. 2023, pp. 1–10.

[2] F. McKee and D. Noever, "Chatbots in a honeypot world," 2023, *arXiv:2301.03771*.

[3] M. Sladić, V. Valeros, C. Catania, and S. Garcia, "LLM in the shell: Generative honeypots," 2023, *arXiv:2309.00155*.

[4] Z. Wang, J. You, H. Wang, T. Yuan, S. Lv, Y. Wang, and L. Sun, "HoneyGPT: Breaking the trilemma in terminal honeypots with large language model," 2024, *arXiv:2406.01882*.

[5] P. P. Ray, "ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," *Internet Things Cyber-Phys. Syst.*, vol. 3, pp. 121–154, Jan. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S266734522300024X

[6] S. Biswas, "Role of ChatGPT in computer programming," *Mesopotamian J. Comput. Sci.*, vol. 2023, pp. 9–15, Feb. 2023. [Online]. Available: https://mesopotamian.press/journals/index.php/cs/article/view/51

[7] M. Zhang and J. Li, "A commentary of GPT-3 in MIT technology review 2021," *Fundam. Res.*, vol. 1, no. 6, pp. 831–833, Nov. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667325821002193

[8] T. B. Brown, "Language models are few-shot learners," 2020, *arXiv:2005.14165*.

[9] Y. Hu, S. Cheng, Y. Ma, S. Chen, F. Xiao, and Q. Zheng, "MySQL-pot: A LLM-based honeypot for MySQL threat protection," in *Proc. 9th Int. Conf. Big Data Analytics (ICBDA)*, vol. 45, Mar. 2024, pp. 227–232.

[10] V. S. Mfogo, A. Zemkoho, L. Njilla, M. Nkenlifack, and C. Kamhoua, "AIIPot: Adaptive intelligent-interaction honeypot for IoT devices," in *Proc. IEEE 34th Annu. Int. Symp. Pers., Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2023, pp. 1–6.

[11] C. Vasilatos, D. J. Mahboobeh, H. Lamri, M. Alam, and M. Maniatakos, "LLMPot: Automated LLM-based industrial protocol and physical process emulation for ICS honeypots," 2024, *arXiv:2405.05999*.

[12] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 36, 2024.

[13] J. Ragsdale and R. V. Boppana, "On designing low-risk honeypots using generative pre-trained transformer models with curated inputs," *IEEE Access*, vol. 11, pp. 117528–117545, 2023.

[14] X. Victoria Lin, C. Wang, L. Zettlemoyer, and M. D. Ernst, "NL2Bash: A corpus and semantic parser for natural language interface to the Linux operating system," 2018, *arXiv:1802.08979*.

[15] M. Knöchel and S. Wefel, "Analysing attackers and intrusions on a high-interaction honeypot system," in *Proc. 27th Asia–Pacific Conf. Commun. (APCC)*, Oct. 2022, pp. 433–438.

[16] M. Schröder and J. Cito, "An empirical investigation of command-line customization," *Empirical Softw. Eng.*, vol. 27, no. 2, p. 30, Mar. 2022.

[17] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," 2019, *arXiv:1908.10084*.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1, J. Burstein, C. Doran, and T. Solorio, Eds., Minneapolis, MI, USA: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[19] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a 'Siamese' time delay neural network," in *Proc. 6th Int. Conf. Neural Inf. Process. Syst.* San Francisco, CA, USA: Morgan Kaufmann, 1993, pp. 737–744.
[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 26, 2013.
[21] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 1, Jun. 2005, pp. 539–546.
[22] J. Xu, R. Zhang, C. Guo, W. Hu, Z. Liu, F. Wu, Y. Feng, S. Sun, C. Shao, and Y. Guo, "vTensor: Flexible virtual tensor management for efficient LLM serving," 2024, *arXiv:2407.15309*.
[23] X. Ning, Z. Lin, Z. Zhou, H. Yang, and Y. Wang, "Skeleton-of-thought: Large language models can do parallel decoding," 2023, *arXiv:2307.15337*.
[24] H. Jiang, Q. Wu, X. Luo, D. Li, C.-Y. Lin, Y. Yang, and L. Qiu, "LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression," 2023, *arXiv:2310.06839*.

SIMON B. WEBER received the M.S. degree in computer science from Heinrich-Heine-Universität Düsseldorf, in 2020, where he is currently pursuing the Ph.D. degree. His research interests include IT and network security, with a particular interest in attack detection for critical infrastructure. Alongside his Ph.D. studies, he is a member of the IT Security Team, University Computing Center.

MARC FEGER is currently pursuing the Ph.D. degree in computer science, with a focus on using machine learning to analyze and extract arguments from texts. He is a Research Associate with Heinrich-Heine-Universität Düsseldorf, working on the "AI for Everyone" project. His work also includes developing argumentation and recommender systems, with a particular emphasis on applying AI to support argumentation in social media. His research interests include argument mining and natural language processing.

MICHAEL PILGERMANN received the Ph.D. degree in information security from the University of South Wales, in 2006. After, he gained 15 years of professional IT security experience in business and administration. Since 2021, he has been a Professor with the Brandenburg University of Applied Sciences, specializing in IT security. His research interest includes the detection of cyber-attacks when operating critical infrastructures.