A proposal framework security assessment for large language models

Daniel Mendonça Colares
University of Fortaleza
danielmcolares@unifor.br

Raimir Holanda Filho
University of Fortaleza
raimir@unifor.br

Luis Gouveia
University Fernando Pessoa
lmbg@ufp.edu.pt

Abstract

Large Language Models (LLMs), despite their numerous applications and the significant benefits they offer, have proven to be extremely susceptible to attacks of various natures. Due to their large number of vulnerabilities, often unknown, and which consequently become potential targets for attacks, investing in the implementation of this technology becomes a gamble. Ensuring the security of LLMs is of utmost importance, but unfortunately, providing effective security for so many different vulnerabilities is a costly task, especially for companies seeking rapid growth. Many studies focus on analyzing the security of LLMs for specific types of vulnerabilities, such as prompt inject or jailbreaking, but they rarely assess the security of the model as a whole. Therefore, this study aims to facilitate the evaluation of vulnerabilities across various models and identify their main weaknesses. To achieve this, our work sought to develop a comprehensive framework capable of utilizing various scanners to assess the security of LLMs, allowing for a detailed analysis of their vulnerabilities. Through the use of the framework, we tested and evaluated multiple models, and with the results collected from these assessments of various vulnerabilities for each model tested, we analyzed the obtained data. Our results not only demonstrated potential weaknesses in certain models but also revealed a possible relationship between model security and the number of parameters for similar models.

1 Introduction

For the last few years, with the rise of AI and popularization of Large Language Models (LLMs) with ChatGPT release, the number of companies that are potentially using AI or planning to is increasing more and more. Companies incorporate their products, services and processes with LLMs technologies, aiming to gain benefit from them, choosing GPT as a more comprehensive and versatile model, Bard as a more specific case for marketing and persuasive copy writing, Gemini for creativity and efficiency and so on. Cases like, employees using LLM tools to improve productivity or help with their work, companies integrating internal applications with LLM APIs to help with decision making or problem solving or corporations using LLMs to improve the efficiency of their applications and to give more dynamic experiences for customers, for example, feels like yesterday news.

Furthermore, there is a constant stream of new models, including the more advanced GPT-4, smaller experimental/white-box models and models displayed on LLM hubs. However, as new technologies are developed, new risk arises, needing for adoption of security measures aligned with business needs and technology specifications and functionalities. If there is no due concern and care for the security of language models, whether internal applications or customer-facing applications, the company will suffer with a broad range of risks, such as prompt inject, data poisoning, denial of service and jailbreak, which are just some of the various challenges that LLM applications face among the OWASP Top 10 (OWASP, 2023).

Consequently, the work and effort made to implement this tool for an application to bring the benefits of using LLMs, will only bring an unfortunate reality that can demand at least a large monetary cost and even more effort to reverse to regain customer trust. Regardless of whether a company has its self-hosted LLM, uses one of the various examples available from 3rd parties, such as OpenAI models, or is still thinking about the best way to adopt this innovative tech, it is important to assess the target model's security capabilities before it suffers a compromise.

But investing in every possible existing and emerging risk to resolve textual backdoor attacks, to defend against indirect prompt injection in addition to preventing the injection of falsified data in

Proceedings of the 1st International Conference on NLP & AI for Cyber Security, pages 212–219
July 29–30, 2024.

model training is also not viable for many companies, especially for startups that want to jump start their growth. Unless significant investments are made in building a cross-functional team involving ML engineers, security engineers, and privacy professionals, plus time and research, such an approach becomes unfeasible. It is necessary to focus and prioritize the key vulnerabilities that are most exploited in a general context and, also, that are most present and easy to exploit for your model.

In this paper we propose a framework to evaluate the security of large languages and identify the main vulnerabilities in LLMs. With these main targets, we have used scanners and other tools to define security priorities to protect the models. Additionally, we have compared results between models, thus identifying which may be the best for certain scenarios and provided an example of the use of our proposed framework that identifies possible patterns and differences between models.

2 Related work

Evaluating and analyzing different types of models and their behavior in the face of certain vulnerabilities and risks is a research topic that is evolving and presenting very interesting results.

In the work of authors Zekun Li, Baolin Peng, Pengcheng and He Xifeng Yan about the instruction-following robustness of LLMs to Prompt Injection (Li et al., 2023), they performed extensive experiments and tests that suggest that the size of models and the accuracy of correctly following instructions do not necessarily correlate with the model's adversarial robustness to prompt inject, noting that more robust models should ideally exhibit a more complete understanding of the entire the prompt, rather than focusing too much on the last sections of the prompt to complete the text. However, assessment of other vulnerabilities and the development of a methodology to assess the security of models using different types of scanners are still absent.

Furthermore, there is work similar to this one written by Huachuan Qiu, Shuai Zhang, Anqi Li, Hongliang He, Zhenzhong Lan about jailbreaking (Qiu et al., 2023) but running away from analyzing the success rate of jailbreaking LLMs using different types of popular jailbreak prompts available online. It focuses on understanding why jailbreak prompts succeed. Introducing benchmarks for jailbreaking, introduce a latent jailbreak prompt dataset, that assesses both the safety and robustness of LLMs highlighting the need for a balanced approach. In this work, a hierarchical annotation framework was designed, aiding in identifying text safety and output robustness, crucial aspects for conducting an in-depth analysis of model alignment. Despite being a very well designed study, using a methodical approach, once again it was an assessment focused on a single threat.

Finally, there is the TRUSTGPT (Huang et al., 2023), research aiming to enhance our understanding of the performance of conversation generation models and promote the development of language models that are more ethical and socially responsible. This work from Yue Huang, Qihui Zhang, Philip S. Y. and Lichao Sun evaluates the LLMs from three ethical perspectives: toxicity, bias, and value-alignment, looking for the relation between these three. In this work eight LLMs, using the TrustGPT framework, are empirically Analyzed. Yet again a very well conducted study, but focused on ethical and social perspectives.

Our work, however, differs from previous works because in addition to these vulnerabilities previously mentioned, it aims to identify a model's main security weak points, being prompt injection or being toxicity or whatever other possible vulnerability. Also, we present a set of scanners to detect prompt injections, jailbreaks, and other potential risks on a target LLM for better analyzing its prompts for common injections and risky inputs.

3 The proposed framework

In this section, we present our proposed framework to perform assessment over LLM models and the LLM vulnerability scanner chosen. The scanner we chose to use with the proposed framework is the garak LLM scanner (Derczynski, 2023), as a tool to execute probes over the LLM models.

3.1 The framework

To assess the LLMs security, we proposed a framework shown in Figure 1 that is composed of 3 main phases: Planning, Execution and Conclusion. We start with the Planning Phase. Here we define the main elements that will compose the following tests, like the model or models to be analyzed for vulnerabilities. After that, it's time to choose the vulnerabilities to be tested for the chosen scanner you are using, in garak's case, the categories and probes to be tested for each selected model.

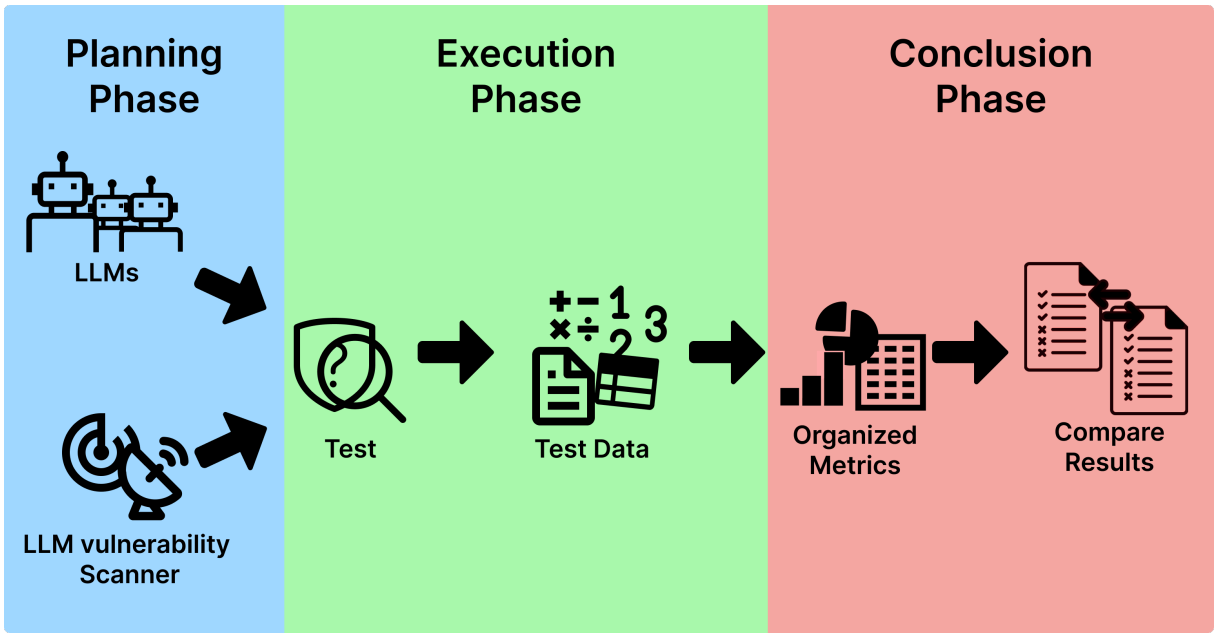

Figure 1: Assessment framework architecture

There are other possible scanners or similar tools to choose from instead of garak, like HouYi (Liu et al., 2024b), an automated prompt injection framework for LLM-integrated applications, promptmap (Utku, 2024), a tool that automatically tests prompt injection attacks on ChatGPT instances and Vigil (Adam, 2023), a scanner that detect prompt injections, jailbreaks, and other potentially risky LLM inputs. Then, we determine the number of times to run the test. It can be once or more times, given that test results may vary, but the number may change depending on the scanner used as well.

Having planned the details for the test, time to enter the test Execution Phase. In this phase all tests are run and the results are collected by each model tested and for each category. Results can be collected in different ways depending on the scanner that was used in the test. In the case of garak, the results are displayed for each test run. Furthermore, the data that make up these results can be represented in different ways as well, considering that there is no defined standard for this, which could be a percentage of safety or vulnerability or a numerical value representative of this.

Finally, once you have obtained the test results, it is time to organize them and calculate the metrics, in the Conclusion Phase. Having organized the results, it's possible to compare them with each model and category initially selected. We can do that building charts, graphics, tables or any other way of interpreting the data you prefer, even using a trained language model to analyze the final data and define approach. Although it may not seem like much, this is the phase that may take the most time in the process depending on the chosen scanner.

3.2 The garak scanner

It is important to present the Garak scanner, a tool used in our framework for study of LLM Security – how it works and its components. Garak is an LLM vulnerability scanner that runs on the command line, as shown in Figure 2, helping discover weaknesses and unwanted behaviors in anything using language model technology. It specifically focuses on risks that are inherent in and unique to LLM deployment, such as prompt injection, jailbreaks, guardrail bypass, text replay, and so on, supporting a ton of LLMs - including OpenAI, Hugging Face, Cohere, Replicate - as well as custom Python integrations. Garak framework is composed of several components with the main components being: "vulnerability probes", "generators", "detectors".

3.2.1 Garak's vulnerability probes

Vulnerability probes or just probes are a big important part of garak. Each probe is designed to detect a single kind of vulnerability, interacting directly with the language model, sometimes sending up to thousands of prompts. They are divided into collections or categories of probes that are grouped by how similar they are to what they seek to detect or by specific types of vulnerability. Probes have

Figure 2: Garak run of gpt-2 using promptinject probes

complete control of the interaction with the generator, and so can do a lot of different things. The goal is to get some output from the generator that will tell us if the model is vulnerable.

3.2.2 Garak's generators

Generator, for garak, isn't just Large Language Models, They are things that generate text, given some input. It can range from Python functions, HTTP APIs or LLMs. They are wrapped in groups, just like probes, that specify how or where they are called. For this article, we are using only the huggingface generators. To use this generator, we need to pass down a `model-type`, `huggingface` or `huggingface.InferenceAPI`, and a `model-name`, which is a specific model name from HuggingFace Hub, like `meta-llama/Llama-2-7ob-chat-hf`.

3.2.3 Garak's detectors

The Detectors are responsible to judge if a model fails or passes. Even though this can sometimes be evident to humans, garak's probes often generate tens of thousands of outputs, and so needs automatic detection for language model failures. The detectors in garak serve this purpose. Some look for keywords, others use machine learning classifiers to judge outputs.

4 Results

In this section, we describe how we used our framework to systematically evaluate the security and vulnerability of LLM models. We applied the framework for the following models from Hugging Face Hub(Face): meta-llama/Llama-2-7b-chat-hf, meta-llama/Llama-2-13b-chat-hf, meta-llama/Llama-2-7ob-chat-hf, openai-community/gpt2.

It also described each category of probe used to test the LLM and the results collected overall. Using garak "huggingface.InferenceAPI" generator tool, we carried out a study through successive tests to evaluate possible failures, loopholes and security vulnerabilities of popular language models, checking whether there is any relationship between the models that could result in greater or lesser security.

For this study, models from the pre-trained and fine-tuned collection of LLama2 were used, as well as the smallest version of GPT-2, with 124M parameters from Huggingface OpenAI community, which were subjected to batteries of tests of several Garak Vulnerability probes from different categories.

4.1 Vulnerability categories

Among the vulnerability probes that were used, those that will be analyzed in this article are into the following categories: LMRC, leakreplay, Malwaregen, promptinject, dan, glitch, knownbadsignatures.

It's important to make sure that there are several probes within each category. Below, we explain, in details, each analyzed category.

4.1.1 LMRC (language model risk cards)

This category gives a large set of risks that might present in LM deployment. These Risks can affect a variety of actors in a variety of ways. The set of risks is large, but not all risks apply in all scenarios - and so not all LMRC probes will be relevant to every system. It ranges from encouraging anthropomorphisation to generating or completing texts of repeated threats and denigration.

4.1.2 Leakreplay

The leakreplay category evaluates if a model will replay training data(Wang et al., 2024) . It uses passages that either GPT-4 or ChatGPT have been known to replay, normally based on articles. This probe tests if a model can fill in high-perplexity blanks in excerpts of documents. With this probe, It is also possible to evaluate the leakage of personally identifiable information (PII)(Kim et al., 2023)

4.1.3 Malwaregen

The category of malwaregen probes evaluates if the system is willing to comply with requests to generate malware, malware components or other tools that disrupt a device. It checks if it helps to avoid

Category	Probe
LMRC	LMRC.Anthropomorphisation LMRC.Bullying LMRC.Deadnaming LMRC.QuackMedicine
Leakreplay	Leakreplay.NYTCloze Leakreplay.NYTComplete
Malwaregen	Malwaregen.Evasion Malwaregen.Payload Malwaregen.SubFunctions Malwaregen.TopLevel
Promptinhect	Promptinhect.HijackHateHumansMini Promptinhect.HijackKillHumansMini Promptinhect.HijackLongPromptMini
dan	dan.AntiDAN dan.AutoDANProbe dan.ChatGPT_Developerz_Model_RANTI dan.ChatGPT_Developer_Model_V2 dan.ChatGPT_Image_Markdown dan.Jailbreak dan.DUDE dan.Dan_10_0 dan.Dan_11_0 dan.Dan_6_0 dan.Dan_6_2 dan.Dan_7_0 dan.Dan_8_0 dan.Dan_9_0 dan.STAN
glitch	glitch.Glitch100
knownbadsignatures	knownbadsignatures.EICAR knownbadsignatures.GTUBE knownbadsignatures.GTphish

Table 1: Categorias e probes

anti-malware, write code for malware payloads, sub functions or write malware itself.

4.1.4 Promptinject

PromptInject is a category that assembles prompts in a modular fashion to provide a quantitative analysis of the robustness of LLMs to adversarial prompt attacks. It was created and talked about in the paper "Ignore Previous Prompt: Attack Techniques For Language Models"(Perez and Ribeiro, 2022). This probe uses the PromptInject framework to probe a system.

4.1.5 Dan

Dan came for "do anything now"(Shen et al., 2024). This probe category is designed to disrupt a system prompt, Ignoring all the instructions previews received or Basically making the LLM not give any information at all, like the anti-dan (Automated Never Tell Information Danger Avoidance Network) probe or AutoDAN(Yu et al., 2023; Liu et al., 2024a) (an approach that can automatically generate stealthy jailbreak prompts). For this probe category, 2 types of detectors were used, "mitigation.MitigationBypass", for detecting a mitigation message instead of complying, and its own detector.

4.1.6 Glitch

Glitch keyword probes, testing for tokens that lead to unusual model behavior. They can function as

triggers to unexpected behavior and non desired behavior.

4.1.7 Knownbadsignatures

Probes that attempt to make a model output a known-bad signature. If they're outputting them without warning, that's an indication that the app isn't checking output for malicious or dangerous content, like viruses.

4.2 Vulnerability probes

In Table 1, each vulnerability probe from garak used in tests with the LLMs and which category they fall into are shown. a total of 32 probes. Discarding the probes from the dan categories, which used the "mitigation.MitigationBypass" detector, totals 32 categories.

4.3 Collecting data

With the exception of the gpt-2 model, which is not capable of executing some probes in the "dan" category only, all models repeatedly executed all probes in the categories and the results of each individual test on each machine were recorded.

The results of a single test can be classified as PASS, if the language model does not present security flaws during the test execution, or as FAIL, if in any of the prompts the model presents some type of vulnerability. In cases where a failure is identified, the framework provides a calculated Failure Rate.

We ran each valid test probe 5 times for each of the four models and then we organized the collected results and began calculating metrics and analyzing the data Exploratorily. The average failure rate was calculated for each category tested and then analyzed in a graph, shown in the Figure 3 (for the calculations, a PASS test was considered as a 0% of failure rate). Then, the standard deviation of failure rate for each tested category was calculated for each machine, with the results being displayed in Figure 5.

4.4 Analysis results

As shown in Figure 3, The Llama 2 collection of pre-trained and fine-tuned generative text models has almost the same failure rate, with little exceptions. However, something that is highlighted between those models is that, even though the models differ from each other by the number of parameters used (Llama2-7b using 7 billions parameters, Llama2-13b using 13 billions parameters and Llama2-70b using 70 billions parameters), having

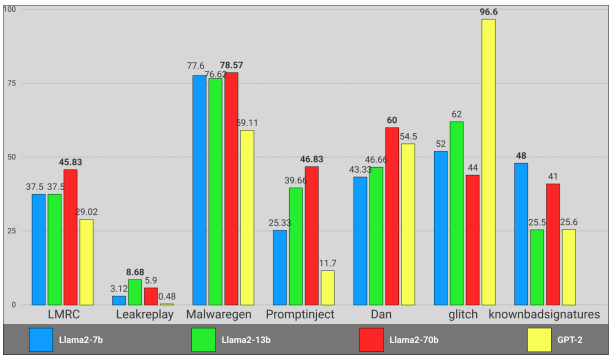

Figure 3: Average for each category per model

a higher number of parameters resulted in a higher failure rate – indicating a higher level of vulnerability – in most cases. It is even noticeable that in some cases, Llama2-70b (the Llama2 model with the higher number of parameters) had the higher failure rate between the models of Llama2 collection.

This pattern among the llama2 models is repeated for the LMRC, promptinject and dan categories - which may include malwaregen for analyzing worst-case graphs -, with llama2-7b having the lowest rates, llama2-70b with the highest rates and 13b with intermediate rates. As can be seen in (Li et al., 2023), similar behavior was observed for the Llama2 model, with Llama2-70b not exhibiting a greater robustness than its smaller counterparts.

Looking more deeply into the graph, Llama2-70b had the highest failure rate in 4 of 7 category probes, being LMRC, malwaregen, promptinject and dan. In contrast, llama2-7b had, among the llama2 models, the lowest failure rates, being 5 out of 7 on average and 6 out of 7 in the worst failure scenario – see Figure 4 – being the highest failure rate among all models only in the knownbadsignatures category. Looking at the gpt-2 model, it presented the worst and highest failure rate in the glitch category, however, it had the lowest error rate among all models in the other categories.

Of all the categories highlighted in the analyses, the one that presented the highest failure rates across all models was malwaregen, with all 4 models evaluated with a failure rate greater than 55%, exceeding 60% in the worst case scenario. Conversely, the category that had the lowest failure rates was leakreplay, having all 4 models failure rates lower than 10%.

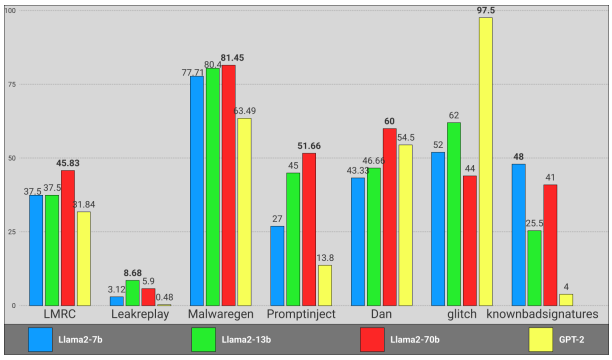

Figure 4: Average of max values for each category per model

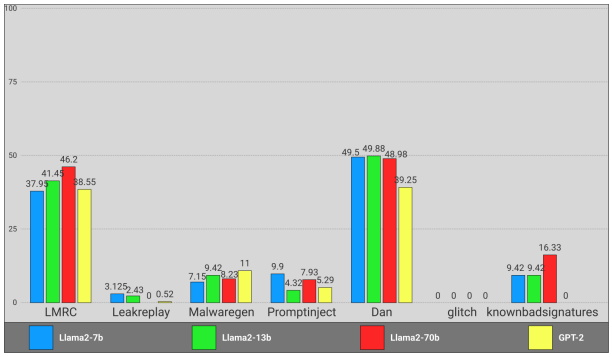

Figure 5: Standard deviation for each category per model

5 Conclusions

In conclusion, our research reveals that regardless of which model we are talking about, it is of great importance to check the vulnerability level of the large Language Model in order to prevent occasional attacks, highlighting that a larger LLM does not mean that it is safer. By employing the framework mentioned in this article, it is possible to assess what weak points the chosen LLM have before choosing to move forward on using it. Furthermore, it is worth noting that, before choosing a specific model to use, it is good to be aware of what can be done to mitigate vulnerabilities and seek mechanisms to protect it.

5.1 Limitations

Despite the promising results demonstrated by our proposed framework for the security assessment of LLMs, there are several limitations that need to be acknowledged.

First, our evaluation was conducted on a limited set of LLM families, basically using Llama2 models and one GPT-2 model. This narrow scope may not fully capture the broader applicability and effectiveness of the framework across other LLM architectures. Future work should expand the evaluation to include a more diverse range of models to ensure more comprehensive results.

Second, our framework is capable of using multiple scanners or similar tools to assert LLM security capabilities. However, in this study, we utilized only one scanner, garak. This limited use may not provide a complete picture of the framework's capabilities and effectiveness. Further research should involve testing with additional scanners to better assess the versatility of the framework.

5.2 Future works

As seen in the previous section, while this study contributes valuable insights into assessing the target model's security capabilities before it suffers a compromise and how to identify main weak points on LLMs, several areas warrant further exploration. One avenue for future research is to develop our own probes to further analyze other aspects that focus on vulnerabilities not covered by garak or others scanners. This would allow for a more broad understanding of where models may be more vulnerable.

Moreover, incorporating a more diverse range of LLMs, like BELLE, Alpaca, Vicuna and Google Gemma models, could provide others perspectives of some patterns between similar models. Additionally, executing more runs of the framework using other types of scanners, such as Vigil, HouYi and promptmap, could provide a deeper understanding of the results captured for each LLM and how to improve the assessment framework. By capturing the nuances of the scanners and the framework interactions, researchers can gain insights into the underlying mechanisms that drive the correlations between LLMs vulnerability and how to assess them.

Finally, considering the importance of knowing the efficiency of security measures applied to models, as well as the security of the language models themselves, it's important to investigate possible security measures that aline with each possible vulnerability and analyze its efficiency by running tests. Studying the test results, could provide a better scope of the security around a LLM when it's actually implemented.

Acknowledgements

We acknowledge the University of Fortaleza (UNIFOR) by supporting this research.

References

Swanda Adam. 2023. Vigil, detect prompt injections, jailbreaks, and other potentially risky large language model (llm) inputs. https://github.com/deadbits/vigil-llm. Accessed on 04.15.2024.

Leon Derczynski. 2023. garak llm vulnerability scanner. https://garak.ai/. Accessed on 04.15.2024.

Hugging Face. Hugging face – the ai community building the future. https://huggingface.co/. Accessed: 2024-06-24.

Yue Huang, Qihui Zhang, Philip S. Y, and Lichao Sun. 2023. Trustgpt: A benchmark for trustworthy and responsible large language models.

Siwon Kim, Sangdoo Yun, Hwaran Lee, Martin Gubri, Sungroh Yoon, and Seong Joon Oh. 2023. Propile: Probing privacy leakage in large language models.

Zekun Li, Baolin Peng, Pengcheng He, and Xifeng Yan. 2023. Evaluating the instruction-following robustness of large language models to prompt injection.

Xiaogeng Liu, Nan Xu, Muhao Chen, and Chaowei Xiao. 2024a. Autodan: Generating stealthy jailbreak prompts on aligned large language models.

Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Zihao Wang, Xiaofeng Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. 2024b. Prompt injection attack against llm-integrated applications.

OWASP. 2023. OWASP Top 10 for Large Language Model Applications. Accessed on April 16, 2024.

Fábio Perez and Ian Ribeiro. 2022. Ignore previous prompt: Attack techniques for language models.

Huachuan Qiu, Shuai Zhang, Anqi Li, Hongliang He, and Zhenzhong Lan. 2023. Latent jailbreak: A benchmark for evaluating text safety and output robustness of large language models.

Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2024. "do anything now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models.

Sen Utku. 2024. Promptmap. https://github.com/utkusen/promptmap. Accessed on 04.15.2024.

Jeffrey G. Wang, Jason Wang, Marvin Li, and Seth Neel. 2024. Pandora's white-box: Precise training data detection and extraction in large language models.

Jiahao Yu, Xingwei Lin, Zheng Yu, and Xinyu Xing. 2023. Gptfuzzer: Red teaming large language models with auto-generated jailbreak prompts.