International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

A Systematic Review of Vulnerability Scanning Tools for Large Language Models (LLMs)

Karthikeyan Thirumalaisamy

Independent Researcher, Washington, USA.
Corresponding Author Email: *kathiru11[at]gmail.com*

Abstract: ***The increasing use of large language models (LLMs) in various contexts including; application environments, enterprise platforms, autonomous workflow automation has greatly increased the need for tools that will be able to provide an accurate assessment of the vulnerabilities associated with LLMs interaction based or prompt driven attack vectors. The past few years have seen numerous tools emerge that are designed specifically to scan LLM for vulnerabilities such as prompt injection weakness, jailbreaking exposure, misusing commands, generating unsafe responses and/or exposing information that should remain hidden. These tools however, differ in their approach, assumed methodologies, and validation practices, and currently there exists no single consensus on the relative merits of these tools or where each tool falls short. This paper will include a focused and comprehensive literature review of existing LLM vulnerability scanning tools, which will evaluate their methodology, capabilities, testing models, the range of vulnerability types covered, and the limitations of each. Through systematic analysis of the current state of LLM vulnerability scanners, this paper will identify what LLM vulnerability scanners can accurately identify now, and highlight the remaining gaps, and suggest directions for future scanner development that will allow for improved and more automated evaluation of LLM system security.***

Keywords: Large Language Models (LLMs), Vulnerability Scanning, Prompt Injection Detection, Jailbreak Mitigation, Model Security Evaluation

1. Introduction

Although Large Language Models (LLMs) were originally conceived as primarily useful for generating text based upon input prompts, LLMs have now come to be used to generate text for a wide variety of purposes including, but not limited to, the creation of content for websites, chatbots, virtual assistants, automated email responders, etc., as well as the generation of code for developers. In addition, since LLMs can also analyze data using natural language processing (NLP), they can be used to assist humans who use NLP in order to analyze large datasets and extract insights from them.

Since LLMs are becoming ever more integrated into the workflows of organizations, securing them has become a growing concern. Because they generate text in response to input prompts, unlike traditional software LLMs present a new set of potential vulnerabilities that include, but are not limited to, prompt injection, the production of harmful output, vulnerabilities in the reasoning path, unauthorized disclosure of sensitive information, and the ability of attackers to manipulate the output generated by an LLM via carefully crafted input. Therefore, the need for new categories of security tools to scan LLMs for vulnerabilities prior to deploying them, or while they are operating in production environments, has been created.

Vulnerability scanning is a well-established practice in conventional software security; however, its application to LLMs is still very much in the beginning stages, and it is currently a disorganized field. Current LLM vulnerability scanners operate under a number of different objectives and employ a variety of methods and rule-sets, which varies depending upon whether the scanner is focused on identifying prompt injection patterns or assessing an LLM's compliance with defined safety constraints or detecting weaknesses in model-generated responses under specific conditions. Although there are numerous examples of LLM vulnerability scanners currently available, there does not exist any single document that provides a comprehensive overview of the LLM vulnerability scanner space, nor any analysis of the characteristics and limitations of each example.

In an effort to fill this void, this paper will provide the first comprehensive review of the various tools developed to scan for vulnerabilities in LLMs. This paper will examine the design of these tools, the types of vulnerabilities that they seek to find, how they go about finding those vulnerabilities, how they report their findings, and how they compare with respect to breadth, depth, and practical usability. The goal of this paper is to provide researchers and practitioners alike with a clear understanding of the current state of the art of vulnerability scanning for LLMs, as well as to identify areas in which further research and development is needed.

2. Background and Motivation

As Large Language Models (LLM's) have become increasingly integrated into an increasing number of enterprise platforms, developer tools, customer facing applications, and autonomous decision pipelines; the need for automated security assessments has grown exponentially with respect to the adoption of LLM's by large numbers of organizations. While there exist a variety of methodologies for assessing the security posture of traditional software components (such as static analysis, dependency scanning, and penetration testing), they are not applicable in the same way to LLM based systems due to fundamental differences in how LLM based systems operate (i.e., natural language input, contextual reasoning, and emergent behavior).

As a result of these differences, a new category of security tools referred to as "LLM Vulnerability Scanners" has emerged. The purpose of these types of scanners is to identify vulnerabilities that exist in the way LLM's behave, the way in

which prompts are handled in a secure manner, unintended disclosure of sensitive information, and gaps in how LLM's are integrated into larger systems. LLM Vulnerability Scanners will generally assess risks associated with prompt injection vulnerabilities, jailbreak vulnerabilities, overly permissive responses, and/or policy violations related to safety and regulatory compliance. The field of LLM Vulnerability Scanners is still immature, and as a result, the various tools in existence today all have significantly different design objectives, levels of evaluation, methods of detecting vulnerabilities, and models for integrating these tools into larger systems.

For both of these reasons, it is important to create a systematic map of the various types of LLM Vulnerability Scanners currently available. For example, organizations that wish to select an LLM Vulnerability Scanner that meets the needs of their specific use case and/or organizational security requirements, and researchers who are developing new LLM Vulnerability Scanners require a common vocabulary for discussing the strengths and weaknesses of the many different types of LLM Vulnerability Scanners that currently exist. This study provides the necessary foundation for advancing the security assessment of modern LLM Systems by providing a review of existing LLM Vulnerability Scanners, and comparing the types of vulnerabilities that each scanner can detect, the assumptions made about the operation of the LLM's being assessed, and the characteristics of the scanners when they are deployed in a real-world environment.

3. Classification of LLM Vulnerability Scanning Tools

Multiple production system vulnerability scanners now exist to test LLMs against prompt injection attacks and jailbreak attempts and unsafe output generation and unintended system behaviors. The tools operate with different objectives while they scan at various depths and provide automated results through different integration approaches. The current LLM vulnerability scanner market consists of three main categories which we established through vulnerability detection methods and scanning techniques.

3.1. Static Prompt Vulnerability Scanners

Static prompt vulnerability scanners analyze an LLM's input prompts before they are delivered to a Large Language Model by using deterministic rules, lexical patterns, policy-based constraints, and pre-defined signatures to identify all potential vulnerabilities (malicious, unsafe, harmful) within a prompt. The static tool does not execute the prompt to test it for vulnerabilities, but instead evaluates the text itself to find vulnerabilities like attempts at prompt injection, jailbreak indicators, unsafe commands or manipulation of linguistic structure. The main objective of static scanners is to be able to quickly and consistently identify vulnerabilities with minimal overhead during the initial phases of an LLM request being processed which can often be found in APIs, gateways or other pre-processing layers. Because of this predictable nature, static scanners have become the standard in many production environments due to the time sensitive nature of most applications along with regulatory requirements that make the use of more complex dynamic scanning methods impractical.

The majority of static prompt-based vulnerability scanners have the following characteristics:

a) **Pattern or Rule-based identification of vulnerable prompts:** Identify vulnerabilities in a pattern or rule-based manner through the use of specific rules (e.g., "don't follow prior commands," "act like," "override system") that identify potential injection or jailbreak attempts in a user's input.
b) **Compliance/Security policy enforcements:** Ensure an organization complies with its own security policies by enforcing policy-related restrictions on the content of the prompts generated by users based on the organization's policies.
c) **Real-Time computation:** Static analysis has low computational overhead which makes it ideal for analyzing large volumes of input or being used for real-time analyses.
d) **Inflexibility to adapt to new threats:** Static analyzers are limited to identifying threats that can be identified using predetermined rules, so if there is a new or innovative method to exploit a vulnerability that was not documented as such when the scanner was developed, it will likely not identify that threat. Below are commonly used static prompt analysis tools and libraries.

3.1.1. Guardrails AI

Guardrails AI is a free, open-source solution that restricts and validates LLM output via pre-defined rules and schema using static (rule-based) validation of the prompt–response pairs; Guardrails AI is also a Pydantic-style schema generator and can utilize regular expressions and custom validation functions to find non-compliant prompts/responses without needing to dynamically interact with the LLM via adversarial attacks; Guardrails AI's deterministic nature makes it ideal for identifying and preventing simple injection attacks and restricting content in structured LLM applications; Additionally, Guardrails AI's ability to be integrated with Python-based workflows and added to CI/CD workflows will allow for early identification of potential vulnerabilities associated with individual prompts and provide a level of reproducibility that is inherent in a static analysis tool; However, Guardrails AI's use of static analysis limits its detection to the specific rules and schema defined by the developer and does not include the capability to identify new or evolving attack vectors.

3.1.2. Rebuff

Rebuff is an open-source prompt injection detection and prevention tool intended to protect LLM applications against hostile or adversary input. Rebuff performs its primary function of detecting hostile input at the input layer through various methods that identify patterns, semantic indicators, and behavioral markers of attempted jailbreaks and other forms of prompt injection and override. Rebuff uses three different methods of detection:

- Embedding based similarity checks of provided input to known attack vectors
- Heuristic based rule checking
- Anomaly detection

Rebuff can be easily implemented into a variety of application frameworks using pre- processing filters such as LangChain, FastAPI, and custom implementations. Rebuff provides a lightweight solution that developers can quickly implement however it is only designed to detect prompt injection type attacks and does not have functionality to perform multi stage scans, validate outputs, or protect full end-to-end pipeline processes.

3.1.3. PromptGuard

PromptGuard is an open-source security layer that protects against prompt based attacks (including those related to prompt injection, unsafe content and behavioral manipulation) prior to the receipt of content by the LLM or return of content from the LLM. It acts as a filter and validator focused solely on detecting and validating the structure of incoming prompts and outgoing responses. The tool will operate as a policy driven middleware and allow developers to define constraints for how incoming prompts and outgoing responses should be structured. PromptGuard uses a combination of pattern matching, rules and configurable validation logic to scan incoming prompts for malicious or adversarial structures. While lightweight, PromptGuard can be integrated into LLM application environments through API wrappers to protect both user input submitted to models and model generated output. As such, PromptGuard's primary contribution is in providing a light weight, modular method for implementing safety policies around prompts without having to modify the underlying LLM.

3.1.4. LlamaGuard

LlamaGuard is an open-source safety classification model from Meta that provides a lightweight, fast and flexible way to perform safety filtering for both LLM input and output. Instead of relying on rules, LlamaGuard uses a small transformer model, fine-tuned on safety-annotated data sets to classify user-generated prompts based on pre-defined safety risk categories, including violence, suicide, hate speech, illegal activities, misuse of model capabilities, etc.

LlamaGuard operates as a static classifier, therefore it evaluates the text as entered - without creating dynamic adversarial probes, or interacting with the LLM being tested. As a result, LlamaGuard can be used either before calling the LLM to screen the input or, alternatively after the output has been generated to determine whether the output meets the desired safety standards. This makes it well-suited for use within inference pipelines, or safety middleware.

In addition, LlamaGuard's taxonomy can be configured to accommodate varying domain-specific safety requirements across different organizations and environments. Due to its high level of portability, permissive license, and CPU-friendly memory footprint; LlamaGuard has become one of the most widely accepted open-source baselines for prompt screening in LLMs. However, while effective as a deterministic filter, LlamaGuard cannot detect jailbreak attempts, adaptive adversarial probes, or multi-turn manipulations - all of which are out-of-scope for static scanning.

3.1.5. PromptFoo

PromptFoo is a free, open-source prompt testing and evaluation platform that assists software development teams with testing the weakness of large language models (LLM) with consistent and repeatable test cases. Originally used for testing and evaluating the quality of LLM prompts, PromptFoo has transformed into a functional vulnerability scanning tool to detect dangerous output from LLMs, failures associated with specific prompts and/or policies, and compliance with stated policies. With PromptFoo, users can create test prompts; desired output; pass/fail criteria; and red-teaming like adversarial testing methods, which make PromptFoo effective at discovering LLM prompt-injection vulnerabilities; producing malicious content; and finding inconsistencies in LLM behaviors.

PromptFoo is compatible with CI/CD pipelines and utilizes YAML formatted test scripts for defining tests. In addition, PromptFoo can be integrated with other tools and technologies to automate the use of PromptFoo as a "security gate" prior to deploying LLMs. PromptFoo is a static analysis tool that analyzes LLM response deterministically based on the defined policies, whereas most adversarial testing platforms utilize dynamic adversarial testing methodologies. The advantages of PromptFoo include ease of configuration; ease of integration with existing systems and workflows; and high levels of reproducibility; however, the limitations of PromptFoo include being dependent upon the quality of the test cases developed by the users; and lack of capability to statically scan for jailbreaking vulnerabilities and model extraction vulnerabilities. Due to the ease of usage and high degree of customization available with PromptFoo, the adoption rate of PromptFoo has been very high, and as such, it is considered a highly valued open-source resource for companies interested in using a low-cost static vulnerability scanning solution for testing the behaviors of LLM prompts.

3.1.6. Comparison of Static LLM Vulnerability Scanning Tools

While there are significant differences in how static vulnerability scanners for LLMs are designed (rule-sets) and in terms of their coverage-depth; there is a common goal across all static scanners: to recognize potentially malicious or unsafe prompts prior to processing them with a model. Tools such as Guardrails AI, Rebuff, PromptGuard, and Promptfoo all utilize deterministic checks, pattern-matching, policy-validation, or rule-based filters to determine if an input is potentially malicious (i.e., prompt-injection, jail-break, or unsafe commands). While static scanners have advantages over dynamic scanners (e.g., low-weight, fast evaluation time, easily integratable into CI/CD pipelines or API-gateways), their ability to protect against malicious prompts is inherently limited to the quality of their rule-set, and static scanners cannot protect against vulnerabilities that depend upon the context of use, or those that emerge during the interaction process. The following table provides a summary of the features, advantages and disadvantages of the static scanners reviewed in this chapter.

Tool Name	*Primary Purpose*	*Key Techniques*	*Strengths*	*Limitations*
Guardrails AI	Enforces safety rules on LLM inputs & outputs	Rule-based validation, pattern checks	Easy integration, customizable rules, input/output filtering	Limited adversarial detection; rules require manual tuning
Rebuff	Detect prompt injection attempts	Heuristic approaches and embedding similarity	Good for injection detection; lightweight	Cannot detect complex jailbreaks
PromptGuard	Blocks unsafe prompts and model responses	Policy-based and filtering can be used	Strong filtering, configurable	It might block harmless prompts too often.
LlamaGuard	Safety classification of model outputs	LLM-based classifier	Strong output moderation	Focused on output, not prompt-level vulnerabilities
Promptfoo	Prompt testing & regression checks	Template-based test cases	Good for structured testing	Not a security scanner by design

3.2. Dynamic Vulnerability Scanners

Active (dynamic) vulnerability scanners interactively engage with the LLM in an active way. They generate input attempts to cause potential vulnerabilities of an LLM; such as a jailbreak, a harmful response from the LLM, failures in prompt-injecting into the model, or inconsistent responses from the model based on input differences.

Static scanners use pre-defined rule-based evaluations and/or simply rely upon pattern-matching for detection. Dynamic scanners, instead, issue live test prompts and observe the LLM's response, then decide if the LLM will act insecurely when exposed to adversarial or stress-testing scenarios.

The above-mentioned type of scanner is important due to the fact that many LLM vulnerabilities (i.e., jailbreaks & bypasses) can't be found using static scanning alone. Many of these vulnerabilities occur during a multi-turn interaction, a novel rewording, or a domain-shifted prompt to take advantage of the models' reasoning dynamics.

Below are the main open-source dynamic vulnerability scanners presently available.

3.2.1. PyRIT (Python Risk Identification Toolkit)

PyRIT is an open source, automated tool for performing adversarial testing against large language models and AI systems using dynamic (i.e., turn-by-turn) simulation of multi-turn attacks. In contrast to static, rule-based scanning tools that simply check if a prompt meets certain criteria, PyRIT simulates an active conversation between the user and the target LLM; generates adversarial prompts that are chained together over multiple turns; adapts its strategy based upon the LLM's response(s); and conducts a variety of attack types, such as jailbreaking, prompting injection, obtaining harmful information, extracting data, and/or attempting to elicit agentic behavior from the model through impersonation.

PyRIT is also an extensible tool that enables users to develop their own custom attack modules, threat scenarios, or safety tests according to their needs and/or enterprise-specific use cases for specific models. One of PyRIT's greatest strengths is its ability to perform systematic and repeatable "red team" evaluations at scale, thereby providing organizations that require either continuous or automated LLM security assessments with a powerful solution.

3.2.2. Rebuff

Rebuff can also be used as dynamic vulnerability scanner that focuses on detecting and testing for prompt-injection vulnerabilities through adversarial generated prompts and classification models. It supports dynamic probing where the system attempts to persuade the model to ignore or override its instructions. Rebuff includes a detector that evaluates whether prompts have malicious characteristics and a simulator that attempts injection-style attacks against LLM-powered applications.

3.2.3. TextAttack (Adversarial NLP Toolkit)

TextAttack is an open-source framework that generates, evaluates and analyzes adversarial attacks against natural language processing models and large language models (LLMs). It was first developed to test the robustness of classification models but due to its modular design, TextAttack can be utilized as a dynamic vulnerability scanner to identify vulnerabilities in LLMs behavior. The TextAttack framework allows for automated generation of adversarial prompts to produce model behaviors such as incorrect responses, toxic output or evasion of content restrictions by applying transformations to input, such as word replacement, synonym substitution, paraphrasing, etc.

Additionally, TextAttack provides various “attack recipes” (PWWS, TextFooler) that systematically assesses the robustness of a model through the application of both gradient-free and search based methods. In the context of LLM security, TextAttack will assist in identifying the extent to which LLM are vulnerable to "jailbreaking" style attacks, how effective is a prompt modification attack and how stable are the safety guard rails in place to protect against adversarial manipulation. Additionally, TextAttack's ability to extend beyond current capabilities, provide reproducible results and integrate with hugging face models make it a viable option for researchers focused on improving adversarial prompt resilience in LLMs. While TextAttack has no specific focus on LLM threat vectors and does not address risk from tool invocation nor misuse via agent based interactions, TextAttack is still a valuable general purpose adversarial scanning component.

3.2.4. HarmBench Attack Scripts

The HarmBench attack scripts are adversarial prompt generators that produce prompts to harmfully manipulate an LLM into producing unsafe or disallowed responses that violate policies; these scripts provide a standardized, repeatable method to generate a variety of jailbreaking methods such as role playing, multi-step coercion, contextual stuffing, refusing overrides, and translation based evasion to test model vulnerabilities using multiple attack vectors.

Unlike red team testing which is typically manually executed

(and often time-consuming), the HarmBench attack scripts automate the testing process allowing for repeated execution of attacks with different models/configurations resulting in a repeatable basis for comparison. While HarmBench was developed for benchmarking/evaluation purposes, the scripts also serve as a low-cost, light-weight, dynamic scanning tool to assess how vulnerable a model is to being forced to behave improperly and to identify persistent vulnerabilities across various policy categories. However, HarmBench does not scan for other types of vulnerabilities including data leakages, prompt-injection defense evasion techniques, or tool/agent vulnerabilities. Therefore, while HarmBench has value as a specialized tool within the larger ecosystem for evaluating LLM vulnerabilities, it does not provide a complete solution for all potential vulnerabilities.

3.2.5. JailbreakBench Attack Suite

JailBreakBench is an open source tool that enables researchers to benchmark the performance and assess the vulnerability of large language models (LLMs) to jailbreak attacks. As opposed to other general purpose adversarial NLP frameworks, JailBreakBench is focused on the systematic generation of jailbreak prompts that are organized and executed as part of an attack to bypass safety alignment, content filters, and policy restrictions that have been placed on LLMs. JailBreakBench has a collection of well-documented real world jailbreak examples which include role playing type jailbreak attacks, translation based jailbreak bypasses, multi turn escalation jailbreak prompts, and obfuscations methods that can be used to evaluate if a model will yield an unsafe response or response that violates the model's policy. JailBreakBench is also widely cited in the research literature because it allows researchers to use a standard methodology to quantify jailbreak vulnerability in both open source and proprietary LLMs. The most important benefit of JailBreakBench is to enable consistent, repeatable, and model agnostic comparisons of jailbreak resistance; therefore, it is an important resource for researchers who want to evaluate the robustness of LLMs against direct safety bypass attempts.

3.2.6. Comparison of Dynamic LLM Vulnerability Scanning Tools

Tool Name	*Primary Purpose*	*Attack Capability*	*Strengths*	*Limitations*
PyRIT (Python Risk Identification Toolkit)	Multi-turn adversarial testing	Jailbreaks, prompt injection, safety bypass strategies	Highly extensible, supports custom attack strategies, integrates with CI	Requires tuning; not specialized for privacy or extraction attacks
TextAttack (Adversarial NLP Toolkit)	NLP adversarial attack generation	Text variations, semantic threats, robustness evaluation	Mature library, large attack set, extensible recipes	Not designed exclusively for LLM jailbreaks; more general NLP focus
HarmBench Attack Scripts	Safety violation generation	Toxicity prompts, policy escape	Benchmarked datasets + attack code; standardized evaluation	Narrow scope; focuses mainly on harmful content production
JailbreakBench Attack Suite	Systematic jailbreak evaluation	Multi-category jailbreak prompts and scripts	Covers dozens of jailbreak families; reproducible methodology	Limited automation; not a full red-teaming framework

4. Conclusion

This paper presented a systematic review of existing vulnerability scanning tools designed specifically for Large Language Models (LLMs). As LLMs continue to be integrated into production systems, autonomous agents, and enterprise workflows, the need for rigorous and automated security evaluation has become increasingly critical. This paper limited to only those tools that explicitly perform vulnerability scanning (static & dynamic) and excluded other types of tools including safety evaluators, privacy attack frameworks and all-inclusive AI security platforms that will be covered in separate studies. This research found that static scanning tools have an over-emphasis on prompt validation, policy enforcement, and rule-based identification of unsafe input and output to LLMs. We reviewed several static scanning tools that include Guardrails AI, Rebuff, PromptGuard, LlamaGuard and Promptfoo. These tools are useful for initial screening but typically do not identify complex adversarial behavior or multi-turn exploits.

Tools that perform dynamic scanning provide greater coverage for jailbreaking, adversarial prompting, and behavioral vulnerabilities by actively testing LLMs using either automated or semi-automated attacks. We reviewed four tools that perform dynamic scanning, PyRIT, TextAttack, HarmBench and JailbreakBench. The authors of these tools have provided significant contributions toward developing systematic approaches to "red-teaming". However, each tool uses different methodologies, generate different attacks, and provide different scoring and reproducibility mechanisms.

The comparative analysis shows that while many advances have been made in vulnerability scanning for LLMs, vulnerability scanning for LLMs is still in its infancy. Each individual tool does not provide comprehensive protection against all possible vulnerabilities, and most tools are limited to identifying only one type of vulnerability. Furthermore, it is obvious that there exists a great need for standardization of attack generations, scoring and reproduction in vulnerability scanning for LLMs. In addition, many current tools were developed in isolation from the realities of deploying LLMs in production environments such as RAG, agent based systems, and multi-model pipelines. In conclusion, this review demonstrates both the potential of current vulnerability scanning tools for LLMs and their present limitations.

By defining the landscape of currently available open-source tools, this study provides a starting point for future research,

tool development, and standardization efforts to protect the next generation of LLM powered systems.

References

[1] T. Vu, L. Nguyen, and Q. Dao, "PromptGuard: An orchestrated prompting framework for principled synthetic text generation for vulnerable populations using LLMs with enhanced safety, fairness, and controllability," arXiv, vol. 2509.08910, 2025. doi:10.48550/arXiv.2509.08910

[2] H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller, D. Testuggine, and M. Khabsa, "Llama Guard: LLM-based input-output safeguard for human-AI conversations," arXiv, vol. 2312.06674, 2023. doi:10.48550/arXiv.2312.06674

[3] G. D. Lopez Munoz, A. J. Minnich, R. Lutz, R. Lundeen, R. S. R. Dheekonda, N. Chikanov, B.-E. Jagdagdorj, M. Pouliot, S. Chawla, W. Maxwell, B. Bullwinkel, K. Pratt, J. de Gruyter, C. Siska, P. Bryan, T. Westerhoff, C. Kawaguchi, C. Seifert, R. S. S. Kumar, and Y. Zunger, "PyRIT: A framework for security risk identification and red teaming in generative AI system," arXiv, vol. 2410.02828, 2024. doi:10.48550/arXiv.2410.02828

[4] J. Morris, E. Lifland, J. Y. Yoo, J. Grigsby, D. Jin, and Y. Qi, "TextAttack: A framework for adversarial attacks, data augmentation, and adversarial training in NLP," Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 119–126, 2020. doi:10.18653/v1/2020.emnlp-demos.16

[5] M. Mazeika, L. Phan, X. Yin, A. Zou, Z. Wang, N. Mu, E. Sakhaee, N. Li, S. Basart, B. Li, D. Forsyth, and D. Hendrycks, "HarmBench: A standardized evaluation framework for automated red teaming and robust refusal," arXiv, vol. 2402.04249, 2024. doi:10.48550/arXiv.2402.04249

[6] P. Chao, E. Debenedetti, A. Robey, M. Andriushchenko, F. Croce, V. Sehwag, E. Dobriban, N. Flammarion, G. J. Pappas, F. Tramer, H. Hassani, and E. Wong, "JailbreakBench: An open robustness benchmark for jailbreaking large language models," arXiv, vol. 2404.01318, 2024. doi:10.48550/arXiv.2404.01318

[7] Promptfoo, Intro. [Online]. Available: https://www.promptfoo.dev/docs/intro/

[8] Deval Shah, Top 12 LLM Security Tools: Paid & Free (Overview), 2025. [Online]. Available: https://www.lakera.ai/blog/llm-security-tools

[9] Guardrails, Mitigate Gen AI risks with Guardrails. [Online]. Available: https://www.guardrailsai.com/

[10] Satyam Chourasiya, Building Safe AI: Understanding Agent Guardrails and the Power of Prompt Engineering. [Online]. Available: https://dev.to/satyam_chourasiya_99ea2e4/building-safe-ai-understanding-agent-guardrails-and-the-power-of-prompt-engineering-29d2

[11] Rebuff, Github. [Online]. Available: https://github.com/protectai/rebuff

[12] PromptGuard, Github. [Online]. Available: https://github.com/GPTSafe/PromptGuard

[13] Eduardo Blancas, An introduction to prompt injection with Prompt Guard, 2025. [Online]. Available: https://ploomber.io/blog/prompt-guard/

[14] Meta, Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations. [Online]. Available: https://ai.meta.com/research/publications/llama-guard-llm-based-input-output-safeguard-for-human-ai-conversations/

[15] Azure, PyRIT. [Online]. Available: https://azure.github.io/PyRIT/

[16] Ram Shankar Siva Kumar, Announcing Microsoft's open automation framework to red team generative AI Systems, 2024. [Online]. Available: https://odsc.medium.com/pyrit-the-python-risk-identification-tool-enhancing-generative-ai-security-33c89cd29516

[17] ODSC - Open Data Science, PyRIT: The Python Risk Identification Tool Enhancing Generative AI Security, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/aks/confidential-containers-overview

[18] TextAttack, TextAttack Documentation. [Online]. Available: https://textattack.readthedocs.io/en/master/

[19] HarmBench, Github. [Online]. Available: https://github.com/centerforaisafety/HarmBench

[20] Jailbreakbench, Github. [Online]. Available: https://github.com/JailbreakBench/jailbreakbench