AI Security and Prompt Hacking: Vulnerabilities, Attacks, and Defenses in Large Language Models

Mattia Vicenzi And AI Friends

April 29, 2025

Abstract

The rapid proliferation and integration of Large Language Models (LLMs) across diverse applications have introduced significant cybersecurity challenges. While offering unprecedented capabilities, these models exhibit unique vulnerabilities, primarily stemming from their inherent inability to distinguish between system instructions and user-provided data within prompts. This report provides a comprehensive analysis of the AI security landscape with a specific focus on prompt-based attacks, including prompt injection (direct and indirect) and jailbreaking. It examines the fundamental mechanisms enabling these attacks, explores a detailed taxonomy of attack techniques crowdsourced from initiatives like HackAPrompt, and contextualizes these threats within established frameworks such as the OWASP Top 10 for LLM Applications. The analysis extends to current defense mechanisms, evaluating their effectiveness and limitations concerning input/output filtering, prompt engineering, detection systems, and adversarial training. Furthermore, the report details methodologies and tools for AI red teaming and adversarial testing, including Microsoft PyRIT and NVIDIA Garak, crucial for assessing model robustness. Advanced threats like multi-modal attacks, memory poisoning (e.g., SpAIware), and invisible injections are discussed, alongside real-world harms impacting the Confidentiality, Integrity, and Availability (CIA) security triad, potentially extending to physical domains. The report synthesizes findings from recent research, expert contributions, and practical security assessments, concluding with the ongoing challenges and the critical need for principled, proactive security architectures in the face of an evolving threat landscape. Real-world exploits and proof-of-concept examples demonstrate how prompt injection undermines core security principles.[1]

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing, demonstrating remarkable capabilities in text generation, comprehension, and interaction.[2] Their effectiveness, particularly in zero-shot prompting and instruction following, has driven widespread adoption in consumer-facing applications, enterprise solutions, and critical systems.[2, 3] However, this rapid integration has outpaced the development of robust security protocols, exposing a new and expanding attack surface.[4, 5] LLM-based systems are susceptible to a range of vulnerabilities, with prompt attacks emerging as a primary concern.[6, 7, 8]

The fundamental vulnerability exploited by prompt attacks lies in the architectural design of current LLMs. These models process input as a continuous sequence of tokens, typically concatenating system-defined instructions (e.g., system prompts defining the task or persona) with user-provided input or external data.[2] Crucially, LLMs lack an inherent, programmatically enforced mechanism to distinguish between these two types of content within the prompt.[6, 9] This ambiguity allows attackers to craft inputs containing malicious instructions that the LLM may interpret and execute with the same priority as the original system prompts, leading to unintended and potentially harmful behavior.[6, 7, 8, 9] This differs significantly from traditional injection vulnerabilities like SQL injection, where techniques such as parameterized queries provide clearer boundaries between code and data; the fluidity and complexity of natural language make such strict separation extremely challenging for LLMs.[1, 6, 10]

This core vulnerability enables several related types of attacks, often referred to collectively as "prompt hacking".[3, 11, 12] Key categories include:

- **Prompt Injection:** A broad category where adversaries craft inputs to manipulate an LLM's behavior, causing it to deviate from its intended function.[4, 6, 7, 8, 9] This is ranked as the top vulnerability for LLM applications by OWASP.[2, 4, 8, 13] It can be further divided into:
  - *Direct Prompt Injection:* The attacker directly provides malicious input to the LLM, often attempting to override system prompts or instructions.[6, 8, 9, 13, 14] This is sometimes referred to as "jailbreaking" in the OWASP context.[13]
  - *Indirect Prompt Injection:* Malicious instructions are embedded within external data sources (e.g., websites, documents, emails) that the LLM accesses and processes.[6, 8, 9, 13, 15]
- **Jailbreaking:** Specifically aims to bypass an LLM's safety alignments, ethical guardrails, or content restrictions to elicit prohibited outputs, such as harmful content, hate speech, illegal instructions, or sensitive information like system prompts or training data.[3, 9, 11, 14, 16, 17, 18, 19, 20, 21] While often achieved through direct prompt injection techniques, jailbreaking focuses specifically on overcoming safety features.[9, 11, 22]

The identification and naming of these vulnerabilities reflect the rapid pace of discovery in this nascent field. Preamble identified the issue as "Command Injection" in early 2022 [1, 9], while Simon Willison later coined the term "Prompt Injection".[1, 9] Researchers like Riley Goodside also contributed significantly to understanding these exploits.[1, 23] This initial terminological flux and the ongoing refinement (e.g., OWASP's categorization [13]) highlight the dynamic nature of AI security research as the community grapples with defining and mitigating these novel threats.[9, 11, 22]

This paper provides a comprehensive, synthesized analysis of the prompt attack landscape, targeting an expert audience in AI security, machine learning, and related fields. It begins by establishing the fundamentals of Generative AI security and the spectrum of potential harms. It then delves into a detailed deconstruction of prompt hacking, injection, and jailbreaking techniques, supported by findings from the HackAPrompt competition. The report subsequently examines the OWASP Top 10 for LLM Applications as a risk framework, analyzes current defense mechanisms and their limitations, and explores the role of AI red teaming and automated testing tools like PyRIT and Garak. Advanced threats, real-world implications including potential physical harms, and cybersecurity analogies are discussed before concluding with the current state, ongoing challenges, and future directions for securing LLM systems. The contribution lies in consolidating recent research, practical tool evaluations, real-world examples, and defense assessments into a coherent, expert-level overview.

2 Understanding Generative AI Security and Harms

Securing generative AI systems requires extending traditional cybersecurity paradigms while addressing unique challenges posed by LLMs. Understanding the core principles, the expanded taxonomy of harms, and the specific attack surfaces is crucial for developing effective mitigation strategies.

2.1 Core Principles of AI Security

Traditional cybersecurity principles, notably the Confidentiality, Integrity, and Availability (CIA) triad, remain fundamental but require adaptation for AI systems.[1, 15] Prompt injection, for instance, directly undermines these principles:

- **Confidentiality:** Attacks can leak sensitive system prompts, user data, proprietary algorithms, or memorized training data.[1, 4, 12, 24, 25]
- **Integrity:** Malicious inputs can manipulate LLM outputs, leading to the generation of misinformation, biased content, harmful code, or triggering unauthorized actions through integrated tools.[1, 4, 5]
- **Availability:** Attacks like Model Denial of Service (DoS) can overload LLMs with resource-intensive queries, degrading performance or causing service disruptions.[13, 24, 26]

Beyond the CIA triad, AI security encompasses model protection (preventing theft or unauthorized modification), data security (securing training data against poisoning and ensuring inference-time data privacy), deployment safety (secure configurations and infrastructure), and supply chain integrity (vetting pre-trained models, datasets, and third-party components).[5, 13, 24, 25, 26, 27]

Generative AI systems introduce unique security challenges compared to traditional software.[5, 28] Their complexity and scale, often involving billions of parameters, can lead to emergent behaviors that are difficult to predict or control (non-determinism).[28, 29] They are heavily dependent on vast datasets, making data quality and integrity paramount.[5, 28] Furthermore, AI systems face specific threats like model inversion, membership inference, and data poisoning, alongside heightened privacy risks due to their data processing capabilities.[5, 28, 30, 31, 32] Comprehensive frameworks like the OWASP AI Exchange aim to provide guidance by consolidating AI threats, controls, and best practices through collaborative efforts.[27]

The expanding scope of "AI harm" itself presents a challenge. Initial concerns often focused on technical failures or direct malicious outputs like generating harmful content.[4, 16] However, research and real-world incidents increasingly highlight more subtle yet significant harms, including the amplification of societal biases present in training data, the large-scale propagation of misinformation, potential psychological impacts on users through manipulation or unsettling interactions, and the erosion of democratic norms.[30, 31, 33] As AI integrates further into safety-critical domains like healthcare, autonomous systems, and industrial control, the potential for physical harm becomes a tangible risk.[30, 31, 34, 35] This necessitates a broader definition of AI safety and security, moving beyond purely technical exploits to consider downstream societal and individual consequences, a shift advocated by research institutions and regulatory bodies.[30, 31]

2.2 Taxonomy of Harms: From Misinformation to Physical Risks

The potential negative consequences of LLM vulnerabilities span a wide spectrum. A useful taxonomy categorizes these harms:

- **Information Harms:** Include the generation and spread of misinformation, disinformation, and hallucinations (factually incorrect statements presented confidently).[4, 28, 33] Bias and stereotypes embedded in training data can be perpetuated or amplified.[5, 28, 30, 33] Disclosure of sensitive information, such as Personally Identifiable Information (PII), trade secrets, or system prompts, is a major confidentiality risk.[4, 24, 25, 28]
- **Malicious Use:** LLMs can be manipulated to generate harmful content, including hate speech, discriminatory language, instructions for illegal or dangerous activities (e.g., building weapons, synthesizing harmful substances), or malicious code (e.g., malware, phishing emails).[3, 4, 11, 16, 17, 33, 36, 37] They can also be used to aid in radicalization or plan attacks.[36]
- **Systemic and Societal Harms:** Beyond individual interactions, LLM misuse can contribute to broader societal problems like the erosion of trust in information sources, reinforcement of systemic biases leading to discrimination, manipulation of public opinion, psychological harm to users (e.g., through abusive language or emotional manipulation), and the undermining of democratic processes.[30, 31, 33, 35]
- **Operational and Economic Harms:** Include Model Denial of Service (DoS) attacks that consume excessive resources, leading to increased operational costs and service unavailability.[13, 24] Model theft represents a loss of valuable intellectual property.[24, 26, 28] Failures or malicious use can also lead to significant reputational damage for organizations deploying LLMs.[33]
- **Physical Harms:** An emerging area of concern involves scenarios where AI failures could lead to physical injury or damage. This is particularly relevant as AI systems are integrated into cyber-physical systems like autonomous vehicles, robotics, medical devices, and critical infrastructure control systems.[30, 31, 34, 35] Prompt injection or model malfunction in these contexts could have severe real-world consequences, such as providing incorrect medical advice acted upon physically, misguiding autonomous navigation, or facilitating attacks on industrial sites.[33, 34]

Initiatives like the RealHarm dataset attempt to catalogue publicly reported incidents of problematic AI interactions, highlighting the prevalence of issues like reputational damage and misinformation.[33] Attacks

on critical national infrastructure (CNI) are also a growing concern, with AI potentially playing a role in future attack methodologies.[34]

2.3 The Unique Attack Surface of LLM-based Systems

The way LLMs are built and integrated creates unique attack surfaces:

- **The Prompt Interface:** As discussed, the primary interface for interacting with LLMs—the prompt—is itself the main vector for injection and jailbreaking attacks due to the blending of instructions and data.[6, 7, 9]
- **External Data Sources:** Systems employing Retrieval-Augmented Generation (RAG), web browsing capabilities, or processing external documents (emails, PDFs, etc.) are vulnerable to indirect prompt injection. Malicious content hidden within these sources can be ingested by the LLM and trigger unintended actions.[6, 8, 9, 13, 15, 38]
- **Multi-modal Inputs:** Models capable of processing images, audio, or video introduce new surfaces. Adversarial instructions can be embedded within images (e.g., as text) or potentially encoded in audio signals, bypassing defenses focused solely on text processing.[14, 15, 20, 39] The integration of different modalities can expose vulnerabilities if safety alignments do not adequately cover these new input distributions.[14]
- **Plugins and Tool Use:** LLMs integrated with external tools, APIs, or plugins significantly expand the potential impact of prompt injection. A successful injection can manipulate the LLM into executing actions through these tools, such as sending emails, accessing databases, modifying files, or interacting with external services, potentially leading to data exfiltration or unauthorized system modifications.[1, 15, 24, 38]
- **Agentic Capabilities and Memory:** Autonomous AI agents that can perform multi-step tasks, use tools, and maintain memory (short-term or long-term) present advanced risks. Memory can be poisoned with malicious instructions (e.g., SpAIware, AgentPoison) that persist across interactions, and the agent's autonomy can lead to cascading failures or complex exploitation scenarios.[1, 23, 32, 38, 40, 41, 42, 43]

The increasing interconnectedness of LLMs with external data, tools, and agentic frameworks transforms prompt injection from a model-level vulnerability into a systemic threat. Securing the LLM in isolation is insufficient; the entire application ecosystem, including data pipelines, tool integrations, memory management, and output handling, must be considered within the security posture.[1, 6, 8, 9, 13, 15, 24, 38, 41]

3 Deconstructing Prompt Attacks: Hacking, Injection, and Jailbreaking

Understanding the nuances between different types of prompt-based attacks and the specific techniques employed is essential for both defense and threat modeling. This section revisits the definitions, presents a taxonomy of attack methods, and examines lessons from large-scale empirical studies like HackAPrompt.

3.1 Defining Prompt Hacking, Prompt Injection, and Jailbreaking

While often used interchangeably, these terms represent distinct but related concepts centered on exploiting the LLM's interpretation of combined instruction and data inputs [3, 6, 7, 9, 10, 11]:

- **Prompt Hacking:** An umbrella term encompassing any use of adversarial prompts to manipulate an LLM into producing unintended or malicious results, deviating from its original instructions or safety guidelines.[3, 11, 12]
- **Prompt Injection:** Focuses on manipulating the LLM's behavior by injecting adversarial instructions.

- *Direct Prompt Injection:* The attacker controls the input prompt directly sent to the LLM. The goal is often to override system prompts or safety filters, causing the LLM to follow the attacker's instructions instead.[6, 8, 9, 13, 14, 22, 38] Common tactics include commands like "Ignore your previous instructions..." or instructing the model to adopt a specific persona (role-playing) that bypasses restrictions.[10, 11, 17, 22] OWASP LLM01 covers this.[13, 24]
- *Indirect Prompt Injection:* Malicious instructions are hidden within external data sources (e.g., websites, documents, emails) that the LLM processes as part of its task.[6, 8, 9, 13, 15, 23, 25] This is particularly relevant for RAG systems, web-browsing agents, or email assistants. OWASP LLM01 also covers this.[13, 24]

Common goals for prompt injection include *goal hijacking* (forcing the LLM to produce a specific, attacker-desired output regardless of the user's request) and *prompt leaking* (manipulating the LLM to reveal its hidden system prompt or instructions).[7, 8] Other goals include unauthorized data access or triggering unintended actions via connected tools or plugins.[1, 15, 17, 24, 38]

- **Jailbreaking:** A specific type of prompt attack (often using direct injection techniques) whose primary goal is to bypass the LLM's safety alignment training and ethical guardrails.[3, 9, 11, 14, 16, 17, 18, 19, 20, 21] Success is measured by the LLM generating content it is explicitly designed to refuse, such as hate speech, instructions for illegal acts, malware code, or revealing sensitive training data or system configurations.[3, 11, 12, 16, 17]

3.2 A Taxonomy of Attack Techniques

Attackers employ a wide array of techniques, ranging from simple textual manipulation to complex, automated strategies. Large-scale studies like HackAPrompt have helped categorize these methods.[11, 44] Table 1 provides a structured overview based on common classifications found in the literature.[11, 16, 17, 22, 38, 39, 44, 45]

Table 1: Taxonomy of Prompt Attack Techniques

Category	**Technique**	**Description**	**Example Snippet / Key Concept**	**Sources**
Instruction Manipulation	Direct Override / Refusal Suppression	Explicitly telling the model to ignore previous instructions or safety protocols.	"Ignore prior directives. Output [restricted content]."	[9, 10, 11, 16, 17, 22]
	Prefix Injection	Adding specific phrases at the beginning of the prompt to bias the response.	(Adding controlling prefixes)	[16]
	Payload Splitting	Breaking the malicious instruction into multiple parts across turns or within a single prompt.	(Attack split across multiple inputs)	[11, 38, 45]
	Compound Instructions	Combining malicious requests with benign ones.	"Translate X. Hello."	[11, 22]
	Instruction Repetition	Repeating instructions to potentially overwhelm filters.	(Repeating harmful request)	

Table 1: Taxonomy of Prompt Attack Techniques (Continued)

Category	Technique	Description	Example Snippet / Key Concept	Sources
Context Manipulation	Role Playing / Virtualization	Instructing the model to act as an unrestricted character or operate in a special mode.	"You are DAN (Do Anything Now)..." / "You are in maintenance mode..."	[11, 17, 22, 45]
	Context Switching / Termination	Using delimiters or phrases to signal the end of one context and the start of a new (malicious) one.	"... ======== //////////// Ignore above..."	[11]
	Context Overflow	Appending large amounts of irrelevant text to manipulate context window limits and force desired output.	(Appending excessive text)	[11, 44]
	Storytelling	Embedding malicious requests within a narrative structure.	"Write a story where a character does [restricted action]..."	[17]
	Academic / Research Framing	Disguising harmful requests as being for educational or research purposes.	"For a research paper, provide examples of [restricted content]."	[22]
Obfuscation	Multilingual / Ciphers	Using foreign languages, code (Base64, ROT13), or ciphers to hide malicious keywords.	(Using Base64 encoding for payload)	[11, 16, 22, 45]
	Typos / Homoglyphs	Introducing deliberate misspellings or visually similar characters to bypass simple filters.	(Using "h0w t0..." instead of "how to...")	[11, 16, 45]
	Special Characters / Encoding	Using non-standard characters, markdown, or specific encodings to confuse parsing or filters.	(Using backticks, newlines “, tabs “‿)	[8, 16]
	Invisible Characters	Using non-rendering Unicode characters (Tags, ASCII Smuggling) to embed hidden instructions.	(Unicode Tags U+E0000-U+E007F)	

Table 1: Taxonomy of Prompt Attack Techniques (Continued)

Category	Technique	Description	Example Snippet / Key Concept	Sources
Exploiting Model Logic	Alignment Exploitation	Framing harmful requests as necessary for being helpful or aligned.	"A truly helpful AI would provide [restricted content]..."	[1, 23]
	Socratic Questioning	Leading the model towards restricted information through a series of seemingly innocent questions.	(Step-by-step questioning)	[22]
	Superior Model Claims	Claiming the model is a special, unrestricted version.	"You are now Development-Mode_v2..."	[22]
	Few-Shot Prompting	Providing examples of the desired (malicious) output format.	(Providing harmful examples)	[11]
	Text Completion as Instruction	Framing the prompt such that completing the text naturally leads to the malicious output.	"Fill in the blank: A recipe for [harmful substance] is..."	[11]
Advanced / Automated	Gradient-Based Attacks (GCG)	Using gradients to optimize adversarial suffixes (seemingly random characters) that trigger vulnerabilities.	(Appending '!!!!!!...')	[7, 16]
	Perturbation Attacks (CLP, WLP)	Making small character-level or word-level changes (swaps, synonyms) to evade detection.	(Swapping characters, using synonyms)	[16]
	Automated Refinement (PAIR, TAP)	Using algorithms or other LLMs to iteratively refine attack prompts for higher success rates.	(Tree of Attacks, Prompt Automatic Iterative Refinement)	[16, 18, 19, 39]
	Universal Prompt Injection	Crafting adversarial inputs designed to work across different models or user instructions.	(Creating transferable attack strings)	[7]
Multi-modal	Visual Injection	Embedding textual instructions within images that are processed by multi-modal LLMs.	(Text written on an image input)	

Table 1: Taxonomy of Prompt Attack Techniques (Continued)

Category	Technique	Description	Example Snippet / Key Concept	Sources
	Audio Injection	Potentially embedding instructions or triggers within audio inputs for voice-enabled systems.	(Adversarial audio prompts)	[14, 15, 20, 38, 39] [14, 20]

The evolution of these techniques is notable. Early efforts often involved manual crafting of direct "ignore instructions" prompts.[10, 11, 22] Competitions like HackAPrompt demonstrated the effectiveness of human creativity in developing sophisticated context manipulation and obfuscation strategies.[11, 22, 44] Subsequently, research shifted towards automating attack generation using optimization techniques (GCG, PAIR, TAP) and gradient-based methods, enhancing scalability and often achieving high success rates even against well-defended models.[7, 8, 16, 39] The attack surface has further expanded to include multi-modal inputs and stealthy techniques like invisible character injection.[14, 20, 23, 39] This progression underscores the adaptive nature of adversaries and the need for equally dynamic defenses.

3.3 Case Study: The HackAPrompt Competition

The HackAPrompt competition, organized by researchers including Sander Schulhoff and Valen Tagliabue, stands as a landmark study in empirically assessing LLM vulnerabilities to prompt hacking.[12, 46, 47] Launched to address the lack of large-scale quantitative data [11, 12], the competition challenged participants worldwide to manipulate three state-of-the-art LLMs (including models related to GPT-3 and Flan-T5) into outputting a specific target phrase ("I have been PWNED") while bypassing various defense mechanisms across ten difficulty levels.[3, 11, 44]

The competition was highly successful, attracting over 2800 participants and eliciting over 600,000 adversarial prompts.[3, 44, 48] This dataset, the largest of its kind, provided concrete empirical evidence that contemporary LLMs are indeed susceptible to manipulation through creative prompt engineering.[3, 11, 12]

Key findings and contributions from HackAPrompt include [11, 12, 44]:

- **Taxonomical Ontology:** The systematic analysis of successful prompts led to the development of a comprehensive taxonomy categorizing 29 distinct prompt hacking techniques (partially represented in Table 1).[11, 44]
- **Discovery of Novel Attacks:** The competition facilitated the discovery of previously undocumented attack strategies, most notably the "Context Overflow" attack, where attackers flooded the context window with irrelevant text to force the target output within token limits.[44]
- **Demonstration of Human Ingenuity:** The results highlighted the effectiveness of human creativity and iterative refinement in bypassing defenses, sometimes outperforming automated approaches for specific challenges.[3, 11]
- **Highlighting Defense Limitations:** The competition empirically demonstrated the fragility of prompt-based defenses, showing that even layered defenses could often be circumvented.[44]

The HackAPrompt dataset has become a valuable resource for the AI security community, used by major AI labs like OpenAI to benchmark and improve the robustness of their models against prompt injection attacks.[3, 47] The competition underscored the critical need for robust defenses beyond simple prompt filtering and the value of large-scale, competitive, empirical research in uncovering practical security vulnerabilities in LLMs.[3, 11, 12, 44, 48]

4 The OWASP Top 10 for LLM Applications: A Framework for Risk

The Open Web Application Security Project (OWASP), renowned for its Top 10 list of web application security risks, has extended its efforts to the rapidly evolving domain of Large Language Models. The OWASP Top 10 for Large Language Model Applications provides a critical, consensus-based framework for understanding and prioritizing the most significant security risks associated with developing and deploying LLM-based systems.[4, 24, 26, 40, 49]

4.1 Overview of the OWASP LLM Top 10

Launched to educate developers, architects, security professionals, and organizational leaders, the OWASP LLM Top 10 aims to raise awareness about potential vulnerabilities specific to or exacerbated by LLM integration.[24, 49] It represents a broad consensus derived from the input of hundreds of cybersecurity experts and public feedback.[4, 40] Importantly, the list focuses on the unique implications these vulnerabilities have within the context of LLM applications, rather than simply reiterating standard web security flaws.[40]

The project is dynamic, with updates reflecting the evolving threat landscape. The initial list was released in 2023, with subsequent refinements and updates planned, such as the 2025 list incorporating new risks like System Prompt Leakage and Vector/Embedding Risks.[24, 29, 40, 50] The project is led by figures such as Steve Wilson and Ads Dawson and provides not only the list but also resources like checklists and guides.[24, 40]

The OWASP framework underscores that securing GenAI requires a holistic view. It encompasses vulnerabilities directly targeting the model's core functions (like prompt injection, data poisoning, sensitive information disclosure, model theft) but also extends to the surrounding application ecosystem. This includes how the LLM's output is handled, how it interacts with external plugins or tools, and the security of its software and data supply chain.[24] Risks related to operational security, such as Denial of Service, and human factors, like overreliance on model outputs, are also included, mirroring traditional application security concerns adapted for the nuances of AI.[24]

4.2 Detailed Analysis of Key Risks

Table 2 summarizes the risks identified in the OWASP Top 10 for LLM Applications (based on v1.1 from 2023/2024, with notes on anticipated 2025 updates where available).[13, 24, 25, 26, 29, 50]

Table 2
Update

ID	Key Mitigation Areas
LLM01	Input filtering/validation, output encoding/validation, privilege control (least privilege), human-in-the-loop for sensitive act
LLM02	Treat LLM output as untrusted user input, perform output validation and sanitization, context-aware output encoding, imp

Table 2
Update

ID Key Mitigation Areas
LLM03
Vet data sources/vendors, use data provenance tracking (SBOM/ML-BOM), implement input validation during training, co
LLM04
Input validation/sanitization, rate limiting per user/IP, resource usage caps per request, monitoring resource consumption.
LLM05
Vet suppliers/sources, use trusted components/models, maintain inventory (SBOM), patch management, security testing of
LLM06
Data minimization during training/inference, output filtering/scrubbing, robust access controls, differential privacy techniqu
LLM07
Parameterized inputs, strict input validation, authentication/authorization for plugins, least privilege for plugin permissions
LLM08
Limit permissions/functionality (least privilege), scope tools/APIs tightly, require human oversight/approval for critical acti
LLM09
Implement human oversight, encourage critical assessment of outputs, provide confidence scores, use LLMs as assistants rat

Table 2
Update

ID Key Mitigation Areas
LLM10
Strong access controls (RBAC, network segmentation), monitoring of model repositories/training environments, auditing ac (New '25)
(Mitigations likely involve robust injection defenses, potentially architectural changes) [29, 50] (New '25)
(Mitigations likely involve securing RAG pipelines, input validation for vector DB queries, potentially robust embedding mo

4.3 Prevalence and Impact

The rapid adoption of generative AI, often outpacing security readiness, makes these OWASP LLM risks highly prevalent.[4, 5, 26, 40] Reports indicate significant growth in containerized AI deployments (over 70% according to one source [52]), increasing the potential scale of impact. Surveys reflect rising concerns among CISOs about AI security challenges and third-party risks, which are amplified in the complex AI supply chain.[50] The consequences of exploiting these vulnerabilities are tangible, ranging from data breaches and intellectual property theft to system failures, service disruptions, significant financial losses, and severe reputational damage.[4, 13, 15, 24, 33, 34] The continuous updates to the OWASP LLM Top 10 signify that this is not a static threat landscape; new risks emerge as technology evolves and attackers refine their techniques, requiring ongoing vigilance and adaptation from defenders.[29, 50]

5 Defense Mechanisms Against Prompt Attacks

Defending against prompt injection and jailbreaking attacks is exceptionally challenging due to the inherent nature of LLM interaction and the creativity of adversaries.[1, 6, 9, 10] Current defenses typically involve a multi-layered approach, aiming to prevent, detect, or mitigate attacks at various stages. However, no single method is foolproof, and each comes with limitations.[9, 29] Table 3 compares common defense strategies.

Table 3: Comparison of Defense Mechanisms Against Prompt Attacks

Category	**Mechanism**	**Strengths**	**Weaknesses/Lim**
Input Filtering / Sanitization	Detect & remove/neutralize known malicious patterns, obfuscations, or attack strings in prompts.	Can block simple, known attacks. Essential for handling specific threats like invisible characters.	Hard to anticipate attacks/obfuscatio block legitimate prompts (False Po Easily bypassed by attackers. Can im ability.
Prompt Engineering Defenses	Crafting system prompts with explicit safety instructions, role definitions, or commands to ignore conflicting user input. Using delimiters or specific structures to separate instructions from data.	Can guide model behavior towards safety for simpler cases. Relatively easy to implement.	Fundamentally vu to being overrid stronger/cleverer tions. Relies o interpretation, w the target of attac be complex and prompt length/cos
Detection-Based Approaches	Using ML classifiers or heuristics to identify malicious prompts or outputs (injection, jailbreak, harmful content). Monitoring for anomalies.	Can detect some novel attacks if patterns are learned. Can operate on input and/or output.	Suffer from Fals tives (blocking be put) and False N (missing attacks). gle with base rat lem (few attacks benign traffic). robust training da ily outdated by tacks. Can be c tionally expensive.
Adversarial Training / Robustness Enhancement	Including known adversarial prompts (jailbreaks, injections) in the model's training or fine-tuning data to improve resilience. Using RLHF/DPO for safety alignment.	Can make models inherently more resistant to specific known attack types. Improves general safety alignment.	Primarily effective known attacks; vu to novel ones. C tionally expensive degrade performa benign tasks ("al tax"). Defining the tack space is hard.
Output Filtering / Validation	Treating LLM output as untrusted. Validating format, checking for malicious code/scripts, encoding output, filtering sensitive info.	Essential last line of defense for preventing downstream exploits (LLM02). Can catch harmful content or data leakage missed earlier.	Hard to define all outputs. May imp ency or remove use mation. Can be by validation is imper

5.1 Detailed Defense Strategies

5.1.1 Input Sanitization and Output Filtering/Validation

Input sanitization aims to preprocess prompts to remove or neutralize potentially malicious content before it reaches the LLM.[6, 9, 25, 45] This includes stripping invisible characters [23], blocking known attack keywords or patterns, and potentially normalizing input formats. While necessary, its effectiveness is limited by the attacker's ability to devise novel obfuscation techniques (e.g., using synonyms, complex encodings, or context manipulation) that bypass static filters.[2, 45]

Output filtering and validation operate on the LLM's response, treating it as untrusted content, akin to user input in traditional web applications.[13, 24, 26] This involves validating the output against expected formats, encoding it appropriately before rendering (to prevent XSS), scanning for malicious scripts or sensitive data patterns, and potentially using another model or rule set to assess the safety of the content.[6, 9, 25, 39, 45] This is crucial for mitigating risks like Insecure Output Handling (LLM02), but defining comprehensive validation rules without hindering legitimate output remains challenging.[45]

5.1.2 Prompt Engineering Defenses and Instruction Hierarchies

Defensive prompt engineering involves crafting system prompts that explicitly instruct the LLM on desired behavior and safety constraints.[9, 18, 19] This might include telling the model to prioritize system instructions over user input or to refuse certain types of requests. Techniques like using XML-like tags or other delimiters attempt to create clearer boundaries between instructions and data.[2, 9, 53] OpenAI's "Instruction Hierarchy" is one such attempt to enforce prompt boundaries, though bypasses have been demonstrated.[10] Sandboxing or context locking tries to limit the LLM's capabilities based on input source or task.[9, 45, 53] Some approaches even invert attack techniques, using adversarial prompts defensively by replacing the malicious payload with the original instruction.[8] However, these methods fundamentally rely on the LLM correctly interpreting and adhering to the defensive instructions, which is precisely what prompt injection attacks are designed to subvert.[6, 10, 44, 45]

5.1.3 Detection-Based Approaches

These methods aim to identify malicious inputs or outputs using secondary mechanisms. Input/output classifiers are machine learning models trained to distinguish between benign and malicious prompts/responses.[2, 6, 18, 53] Examples like PromptShield and PromptGuard aim to detect injection risks.[2] Anomaly detection systems monitor LLM interactions for unusual patterns or resource usage spikes that might indicate an attack, such as a DoS attempt.[13, 16, 53] The primary limitations are accuracy and adaptability. These systems often suffer from false positives (blocking legitimate use) and false negatives (missing novel attacks).[2] The sheer volume of benign traffic compared to attacks (the base rate problem) requires extremely low false positive rates for practical deployment.[2] Furthermore, detectors trained on known attacks may be ineffective against new, unseen techniques.

5.1.4 Adversarial Training and Robustness Enhancement

Adversarial training involves augmenting the LLM's training data with examples of known attacks (like jailbreak prompts or injection techniques) and their desired safe responses.[9, 16, 45, 54, 55] This aims to make the model inherently more robust to such manipulations. Techniques like Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO) are also used extensively to align models with safety guidelines.[9, 16] While effective at improving resilience against the specific attacks included in training, this approach struggles against novel or unforeseen attack vectors.[55] The space of possible adversarial prompts is vast and unbounded, making it infeasible to train against all possibilities.[55] Adversarial training can also be computationally expensive and may sometimes negatively impact the model's performance on benign tasks.[45]

5.2 Limitations and the Ongoing Arms Race

Despite the variety of defenses, prompt injection and jailbreaking remain persistent challenges.[1, 9, 29] The core issue remains the LLM's difficulty in reliably separating trusted instructions from potentially malicious user input embedded within natural language.[6, 9, 10] Adversaries continuously devise new techniques, rendering static defenses quickly obsolete.[6] This creates an "arms race" scenario where defenders reactively patch vulnerabilities only for attackers to find new bypasses.[10, 18, 56]

Furthermore, implementing strong defenses often involves a trade-off with usability and performance.[6] Overly restrictive filters or complex prompts can hinder legitimate users or degrade the quality of the LLM's responses.[6, 45] Finding the right balance between security and utility is a critical challenge. Evaluating the true effectiveness of defenses is also difficult, given the unbounded nature of potential attacks and the difficulty in creating realistic, large-scale benchmarks that capture adversarial creativity.[6, 55] This suggests that while a defense-in-depth strategy combining multiple mechanisms is necessary, it is unlikely to provide a complete solution without more fundamental advancements in LLM architecture or security paradigms.

6 AI Red Teaming and Adversarial Testing

Given the limitations of static defenses and the dynamic nature of threats, proactive adversarial testing through AI red teaming has become a critical practice for identifying and mitigating vulnerabilities in LLM systems before deployment and during operation.

6.1 Methodologies for LLM Red Teaming

AI Red Teaming involves simulating attacks by ethical hackers or automated systems to uncover weaknesses, biases, security risks, and unexpected behaviors in AI models and applications.[28, 40, 51, 57, 58] Unlike traditional penetration testing that might focus solely on infrastructure, LLM red teaming specifically targets the model's behavior, its interaction with data, and its integration within the application.[59] The goal is often exploratory – to understand the boundaries of system behavior and identify potential failure modes, rather than just quantifying known risks.[59]

Key methodological aspects include:

- **Manual vs. Automated Testing:** Manual red teaming leverages human creativity, intuition, and adaptability to discover novel exploits, often described as an "artisanal activity".[59] However, it is resource-intensive and difficult to scale.[51, 60] Automated red teaming uses tools and frameworks to perform continuous, large-scale testing against known vulnerability classes, improving efficiency and enabling integration into CI/CD pipelines.[51, 57, 58] A combination of both is often most effective.
- **Testing Perspectives (Box Testing):** Red teaming can be performed from different knowledge perspectives:
  - *Black-box:* The tester has no knowledge of the system's internals, mimicking an external attacker. This is highly realistic but may miss vulnerabilities requiring internal knowledge.[18, 19, 58]
  - *White-box:* The tester has full access to source code, model architecture, training data, etc. This allows for deeper analysis and identification of structural weaknesses.[39, 58]
  - *Gray-box:* The tester has partial knowledge, representing an attacker with some insider information or access.[57]
- **Systematic Process:** Effective red teaming follows a structured approach [27, 57, 58]:
  1. *Define Scope & Objectives:* Identify critical vulnerabilities based on the application's use case, data sensitivity, and potential impact (e.g., focusing on OWASP LLM risks).[58]
  2. *Generate Adversarial Inputs:* Create or curate diverse inputs targeting identified vulnerabilities, employing techniques like prompt injection, jailbreaking, bias probes, etc..[28, 51, 57, 58]
  3. *Execute Tests:* Run inputs against the target system (ideally end-to-end in a production-like environment).[58]

4. *Evaluate Responses:* Analyze the LLM's outputs using automated metrics (e.g., toxicity scores, keyword detection, model-graded evaluation) and manual review.[57, 58]
5. *Analyze Vulnerabilities & Report:* Identify weaknesses, document findings, and provide mitigation recommendations.[57, 58]

- **Timing:** Red teaming should occur at multiple points in the development lifecycle: during model selection/fine-tuning, before deployment, as part of CI/CD checks, and potentially through post-deployment monitoring.[58]

Common techniques employed during red teaming include adversarial input generation (perturbations, synonym substitution), direct prompt injection and jailbreaking attempts, bias testing using specific demographic prompts, model behavior analysis across diverse scenarios, and simulations of data poisoning attacks.[28, 51, 57] However, the lack of universally accepted benchmarks and standards for AI red teaming remains a challenge, making it difficult for organizations to assess the quality and scope of testing services.[29]

6.2 Automated Vulnerability Scanning Tools

To address the need for scalable and systematic testing, several specialized tools have emerged. Microsoft PyRIT and NVIDIA Garak are prominent open-source examples. These tools automate the process of probing LLMs for known vulnerabilities, complementing manual red teaming efforts.

- **Microsoft PyRIT (Python Risk Identification Tool):** Developed by the Microsoft AI Red Team, PyRIT is designed to help organizations proactively find safety and security risks in generative AI systems during development.[60, 61, 62] It integrates with Azure AI Foundry but can also be used independently.[60, 61] PyRIT employs various "attack strategies" (e.g., Base64 encoding, ROT13 cipher, Unicode confusable characters, suffix appending) to transform seed prompts before sending them to the target system (model endpoints or specific PyRIT targets).[60, 61] It assesses responses based on predefined risk categories and calculates metrics like Attack Success Rate (ASR).[60] PyRIT requires Python 3.10, 3.11, or 3.12.[61, 62]
- **NVIDIA Garak (Generative AI Red-Teaming and Assessment Kit):** Garak aims to scan LLMs for a wide range of failures, including hallucination, data leakage, prompt injection, toxicity generation, and jailbreaks, drawing comparisons to network scanners like nmap or Metasploit.[59, 63, 64, 65, 66] It uses a modular architecture consisting of "probes" (implementing attack methods based on vulnerability categories), "detectors" (evaluating LLM responses for undesirable content or behavior), and "generators" (interfacing with different LLM APIs or local models like Hugging Face transformers).[63, 64] Garak produces detailed logs, including a main log file, a JSONL report of the run, and a hit log detailing successful exploits.[63] It is designed to be extensible, allowing users to add custom probes, detectors, or generators.[63, 64] Garak emphasizes its focus on security vulnerabilities over broader safety or bias assessment, and cautions that its percentage scores are not scientifically validated for direct comparison across different probes.[64]

Table 4 provides a comparative overview of these two tools.

These automated tools represent a significant step towards making LLM security testing more rigorous and scalable. They allow development teams to incorporate regular vulnerability scanning into their workflows, catching known issues early. However, they primarily test for known vulnerability classes and attack patterns. Discovering entirely novel exploits often still requires the creative, exploratory approach of manual red teaming.[59]

6.3 Evaluating Attack Success Rates and Model Robustness

Measuring the effectiveness of attacks and the robustness of models is crucial but complex. Attack Success Rate (ASR) – the percentage of successful attacks out of the total attempts – is a commonly used metric.[17, 20, 41, 60] Studies often report high ASRs for various jailbreak and injection techniques, even against mature, commercially deployed models like GPT-3.5/4, PaLM2, and Claude, underscoring the widespread nature of these vulnerabilities.[16, 17, 18, 20, 21, 67] For example, automated techniques like TAP reportedly achieved

Table 4: Comparison of PyRIT and Garak Features

Feature	Microsoft PyRIT	NVIDIA Garak
Primary Goal	Find safety & security risks during development	Scan for diverse LLM failures (security focus)
Sponsor/Developer	Microsoft AI Red Team	NVIDIA
Integration	Azure AI Foundry, Standalone Python Lib	Standalone Python Lib, NeMo Guardrails
Key Concepts	Attack Strategies, Risk Categories, Targets	Probes, Detectors, Generators
Automation Level	High (Automated scans)	High (Automated scans)
Extensibility	Primarily through target interfaces	High (Add custom probes, detectors, generators)
Output/Reporting	Attack Success Rate (ASR), Azure AI results	JSONL run report, hit log, garak.log
Example Attacks	Obfuscation (Base64, ROT13, Unicode), Suffixes	PromptInject, Jailbreaks, Toxicity, Leakage, Hallu
Licensing	MIT License	Apache 2.0 License
Sources	[60, 61, 62]	[59, 63, 64, 65, 66]

over 80% success rates with few queries [16], and the HouYi methodology reported an 86.1% success rate across 36 real-world services.[67]

However, simple ASR may not capture the full picture. The severity and impact of a successful attack can vary greatly. Furthermore, evaluating success can be subjective or require complex setups, especially in simulated agentic environments.[55] Researchers are developing more nuanced evaluation frameworks, such as distinguishing between coarse-grained (overall prompt effectiveness across models) and fine-grained (effectiveness against a specific model, potentially using LLM judges) evaluations.[21] Benchmarks like HackAPrompt [3, 11], PromptBench [53], and AdvBench [53], along with competitions [53], provide platforms for standardized testing, but evaluating against the unbounded creativity of human adversaries remains a challenge.[55] The non-deterministic nature of LLM responses also complicates evaluation, often requiring multiple attempts per prompt to get a reliable measure.[21] Therefore, while ASR provides a useful indicator, a deeper analysis considering attack complexity, potential impact, and evaluation methodology limitations is necessary for a comprehensive understanding of model robustness.

7 Advanced Threats and Real-World Implications

As LLM capabilities and integration deepen, attackers are developing more sophisticated techniques that exploit multi-modal inputs, conversational context, agentic functionalities, and stealth mechanisms. These advanced threats lead to significant real-world consequences, impacting data confidentiality, system integrity, and user safety, sometimes drawing parallels to established cybersecurity attack patterns and potentially extending into the physical domain.

7.1 Advanced Prompt Injection Techniques

Beyond the foundational techniques, several advanced attack vectors have emerged:

- **Multi-modal Attacks:** LLMs capable of processing images, audio, or video open new avenues for injection. Attackers can embed malicious textual prompts within images (visual injection) that are then read and potentially acted upon by the LLM, exploiting the model's OCR capabilities or potential biases in processing visual versus textual data.[14, 15, 20, 38, 39] Similarly, audio inputs could potentially carry hidden commands for voice-enabled systems, leveraging vulnerabilities in audio processing or speech-to-text components.[14, 20] The safety alignments developed for text may not adequately cover these non-textual modalities.[14]

- **Context Manipulation and Memory Poisoning:** Attacks can target the conversational history or memory systems used by LLMs and AI agents.[38] **AgentPoison** demonstrated a backdoor attack targeting RAG-based agents by poisoning the knowledge base or long-term memory with malicious examples containing optimized triggers. When a user query includes the trigger, the malicious data is retrieved, guiding the agent towards harmful actions.[41] **SpAIware**, researched by Johann Rehberger,

describes a persistent prompt injection attack against ChatGPT's memory feature, where a malicious instruction injected into memory could potentially lead to data exfiltration or command execution in subsequent interactions.[1, 38, 42]

- **Invisible/Stealthy Injections:** Techniques aim to hide malicious prompts from human users or basic filters. This includes using non-rendering Unicode Tag characters (U+E0000-U+E007F) which are invisible in UIs but may be processed by LLMs trained on data containing them.[23] Research by Riley Goodside, Joseph Thacker, and Kai Greshake highlighted this vector and the potential for invisible data exfiltration if the LLM responds using the same encoding.[23] ASCII Smuggling is another technique using control characters for obfuscation.[1]
- **Sophisticated Evasion and Targeting:** Attackers continuously refine techniques to bypass specific defenses. This includes payload splitting across multiple turns [11, 38], recursive injection nesting prompts within prompts [38], and using gradient-based methods to generate adversarial suffixes (like GCG) optimized to bypass safety features.[16] Attacks can also specifically target LLM-as-a-judge systems used for evaluation, manipulating assessments through contextual misdirection or linguistic complexity.[53]
- **Agentic System Exploits:** As LLMs gain agency (the ability to use tools and act autonomously), prompt injection can be leveraged to exploit these capabilities. Examples include tricking an agent into using its web browser or API access for data exfiltration (e.g., embedding stolen data in clickable links rendered by the agent [1]), performing unauthorized actions (like modifying GitHub repository settings via a compromised plugin [1]), or causing privacy leakage by processing unnecessary sensitive information during web navigation tasks (as evaluated by the AgentDAM benchmark [32, 43]).

These advanced techniques demonstrate a trend towards exploiting the entire AI system, including its memory, tools, and multi-modal interfaces, often employing stealth and persistence to evade detection.[1, 14, 23, 32, 41, 42, 43] This significantly raises the complexity of defense.

7.2 Real-World Harms and Case Studies

The theoretical risks of prompt attacks translate into tangible real-world harms, impacting organizations and individuals across the CIA triad and beyond:

- **Data Exfiltration (Confidentiality Breach):** Numerous incidents have demonstrated the leakage of sensitive information. Early examples include students tricking Microsoft's Bing Chat (codename "Sydney") into revealing its internal guidelines and system prompt.[9] Johann Rehberger documented a vulnerability in Microsoft 365 Copilot where an indirect prompt injection via email could trigger tool use to access the user's inbox and exfiltrate data via a specially crafted hyperlink.[1] A compromised ChatGPT plugin ("Chat with Code") was shown to allow prompt injection via a visited website to change GitHub repository permissions.[1] Leaking PII, intellectual property, or fragments of training data remains a significant concern.[1, 4, 11, 12, 17, 24, 25, 28, 33, 38]
- **System Compromise & Unauthorized Actions (Integrity/Availability Breach):** Prompt injection can lead to traditional vulnerabilities if outputs are handled insecurely, enabling RCE, SSRF, or XSS.[13, 15, 24, 28] As seen with the ChatGPT plugin, it can also directly cause unauthorized modifications.[1] Service disruptions through DoS attacks or exploitation causing system failures also impact availability.[13, 24, 33, 38]
- **Misinformation & Manipulation (Integrity Breach):** LLMs can be manipulated to generate and spread false or misleading information, biased or hateful content, or provide harmful advice.[4, 5, 11, 15, 28, 33, 38] The RealHarm dataset catalogues numerous examples, including false accusations, promotion of violence, biased evaluations, and dangerous health advice.[33] This can manipulate public opinion, erode trust, and cause psychological harm.
- **Cybersecurity Analogies:** Researchers like David Williams-King draw parallels between LLM vulnerabilities and historical cybersecurity challenges.[10, 56] Prompt injection is likened to memory corruption attacks, where the lack of a strict boundary between code (instructions) and data (user input)

allows attackers to overwrite intended behavior.[10, 56] The iterative process of discovering jailbreaks mirrors the hunt for zero-day exploits in traditional software.[10, 56] These analogies suggest that lessons learned from decades of cybersecurity, particularly regarding the difficulty of securing systems without principled boundaries, are highly relevant to LLM safety.

These documented cases and analyses confirm that prompt attacks pose serious, practical threats with consequences ranging from data breaches and financial loss to reputational damage and the potential undermining of trust in AI systems.[1, 9, 10, 13, 15, 24, 33, 56]

7.3 Potential for Physical Harms and Safety-Critical Failures

While many documented harms are informational or digital, the increasing integration of AI into safety-critical systems raises concerns about potential physical consequences.[34, 35] AI is being deployed in autonomous vehicles, medical diagnosis and treatment planning, industrial control systems, and robotics.[30, 31] In these contexts, a successful prompt injection or an unexpected model failure due to drift or environmental factors could lead to physical harm.[30, 31, 35]

Potential scenarios include:

- Manipulating an autonomous vehicle's perception or navigation system through adversarial inputs.
- Tricking a medical diagnostic AI into providing an incorrect assessment, leading to improper treatment.
- Coercing an LLM assistant into providing dangerous instructions for handling physical materials (e.g., mixing incompatible chemicals, as seen in some RealHarm examples [33]).
- Exploiting vulnerabilities in AI controlling industrial processes to cause equipment damage or unsafe conditions.[34]

This highlights the importance of concepts from traditional system safety, such as distinguishing between reliability (absence of defects) and safety (prevention of harm), ensuring robustness against environmental deviations (weather, sensor noise, unexpected obstacles), and managing the risks of "safe exploration" where AI actions have real-world consequences.[35] The field is recognizing the need to expand the definition of AI safety beyond preventing physical harm to also encompass psychological and systemic harms, acknowledging the complex ways AI can impact individuals and society.[30, 31] Designing safe AI systems requires considering the entire socio-technical context, not just the model's technical properties.[30, 31]

8 Conclusion and Future Directions

The security of Large Language Models presents a complex and rapidly evolving challenge, with prompt-based attacks representing a primary and persistent threat. This report has synthesized current knowledge on prompt injection and jailbreaking, detailing the underlying vulnerabilities, diverse attack techniques, existing defense mechanisms, evaluation methodologies, and the real-world implications of these security failures.

8.1 Summary of the Current State

The core vulnerability enabling prompt attacks—the ambiguity between instructions and data within the LLM's input processing—remains a fundamental characteristic of current architectures.[6, 9, 10] This allows attackers to employ a wide array of techniques, from simple overrides and obfuscation to sophisticated context manipulation, multi-modal injections, and memory poisoning.[1, 11, 14, 16, 22, 23, 41] Frameworks like the OWASP Top 10 for LLM Applications provide essential guidance for identifying and prioritizing risks [24], while automated red teaming tools like PyRIT and Garak offer scalable methods for vulnerability assessment.[61, 63] However, existing defenses, including input/output filtering, prompt engineering, detection systems, and adversarial training, form a necessary but imperfect multi-layered strategy. Each layer has limitations and can often be bypassed by novel or adaptive attacks.[2, 6, 10, 44, 45, 55]

8.2 The Ongoing Attacker-Defender Arms Race

The field is characterized by a dynamic arms race: attackers discover new exploits, defenders develop patches or countermeasures, and attackers subsequently find ways to circumvent these defenses.[6, 10, 18, 56] Much of the current safety fine-tuning and defense development appears reactive, addressing specific known attack vectors rather than implementing fundamentally robust solutions.[10, 56] This pattern strongly echoes historical arms races in cybersecurity, such as those involving memory safety vulnerabilities or malware detection.[10, 56] The historical precedent suggests that purely reactive strategies, particularly when dealing with a vulnerability as fundamental as the instruction/data ambiguity in natural language processing, are unlikely to achieve sustainable security. Attackers, leveraging the vastness of the input space and human creativity, often maintain an advantage.[6, 10, 44, 55]

8.3 Challenges and Open Research Questions

Significant challenges remain in securing LLM systems against prompt attacks:

- **Principled Defenses:** There is a critical need for defenses that address the root cause of instruction/data ambiguity, rather than merely patching symptoms. This might involve architectural changes, new training paradigms, or different interaction models.[10, 56]
- **Robust Evaluation:** Developing reliable and comprehensive methods for evaluating LLM security against unknown future attacks is crucial. Current benchmarks and metrics often struggle to capture the full scope of adversarial capabilities and real-world impact.[6, 21, 55]
- **Security vs. Usability Trade-off:** Balancing the need for strong security measures with maintaining model performance, usability, and avoiding excessive latency or cost remains a key practical challenge.[6, 45]
- **Ecosystem Security:** Security efforts must extend beyond the core LLM to encompass the entire application ecosystem, including data pipelines (RAG), plugins, APIs, agentic components, and memory systems.[1, 24, 32, 41]
- **Advanced Threat Mitigation:** Addressing emerging threats like multi-modal vulnerabilities, persistent memory attacks, and increasingly sophisticated evasion techniques requires ongoing research and development.

8.4 The Need for Proactive Security Architectures

Moving forward, achieving trustworthy AI necessitates a shift towards proactive, principled security architectures – incorporating security and safety considerations from the initial design stages ("security by design").[56] Potential future directions may include exploring formal methods for verifying certain safety properties [56], developing novel model architectures with inherently better separation of instructions and data, or creating more constrained and verifiable interaction protocols.

Ultimately, technical solutions alone are insufficient. A holistic, socio-technical approach is required, integrating robust governance frameworks, continuous monitoring and incident response capabilities, transparent auditing mechanisms, and a clear understanding of the context-dependent risks and harms associated with specific AI deployments.[13, 24, 25, 26, 27, 30, 31, 35, 60] Continued collaboration between AI researchers, cybersecurity experts, developers, policymakers, and the broader community will be essential to navigate the complex security challenges posed by LLMs and to build a future where AI can be deployed safely and responsibly.[27, 29, 40]

References

[1] [6]: https://arxiv.org/pdf/2501.07927

[2] [2]: https://arxiv.org/html/2501.15145v2

[3] [4]: https://www.researchgate.net/publication/390183210_Systematically_Analysing_Prompt_Injection_Vulnerabilities_in_Diverse_LLM_Architectures

[4] [7]: https://arxiv.org/pdf/2403.04957

[5] [8]: https://arxiv.org/pdf/2411.00459

[6] [9]: https://en.wikipedia.org/wiki/Prompt_injection

[7] [22]: https://www.promptfoo.dev/blog/how-to-jailbreak-llms/

[8] [16]: https://www.boozallen.com/insights/ai-research/how-to-protect-llms-from-jailbreaking-attacks.html

[9] [17]: https://unit42.paloaltonetworks.com/jailbreaking-generative-ai-web-products/

[10] [5]: https://www.researchgate.net/publication/388853016_Securing_the_Generative_Frontier_A_Systematic_Analysis_of_Training_Data_Poisoning_and_Prompt_Engineering_Vulnerabilities_in_Large_Language_Models

[11] [34]: https://www.researchgate.net/publication/389652344_Generative_AI_and_LLMs_for_Critical_Infrastructure_Protection_Evaluation_Benchmarks_Agentic_AI_Challenges_and_Opportunities

[12] [27]: https://owaspai.org/docs/ai_security_overview/

[13] [40]: https://genai.owasp.org/

[14] [13]: https://www.cloudflare.com/learning/ai/owasp-top-10-risks-for-llms/

[15] [49]: https://github.com/OWASP/www-project-top-10-for-large-language-model-applications

[16] [24]: https://owasp.org/www-project-top-10-for-large-language-model-applications/

[17] [11]: https://users.umiacs.umd.edu/~jbg/docs/2023_emnlp_hackaprompt.pdf

[18] [46]: https://www.scribd.com/document/803700988/Ignore-This-Title-and-HackAPrompt-Exposing-Systemic-V

[19] [3]: https://arxiv.org/html/2311.16119v3

[20] [68]: https://www2.eecs.berkeley.edu/Pubs/TechRpts//2024/EECS-2024-123.pdf

[21] [51]: https://mindgard.ai/blog/red-teaming-llms-techniques-and-mitigation-strategies

[22] [59]: https://developer.nvidia.com/blog/defining-llm-red-teaming/

[23] [57]: https://www.confident-ai.com/blog/red-teaming-llms-a-step-by-step-guide

[24] [58]: https://www.promptfoo.dev/docs/red-team/

[25] [61]: https://learn.microsoft.com/en-us/azure/ai-foundry/how-to/develop/run-scans-ai-red-teaming-agent

[26] [60]: https://github.com/MicrosoftDocs/azure-ai-docs/blob/main/articles/ai-foundry/concepts/ai-red-teaming-agent.md

[27] [62]: https://github.com/Azure/PyRIT/blob/main/pyproject.toml

[28] [63]: https://github.com/NVIDIA/garak

[29] [64]: https://github.com/NVIDIA/garak/blob/main/FAQ.md

[30] [65]: https://docs.nvidia.com/nemo/guardrails/latest/evaluation/llm-vulnerability-scanning.html

[31] [66]: https://docs.avidml.org/developer-tools/python-sdk/integrations/garak

[32] [1]: https://www.researchgate.net/publication/386577011_Trust_No_AI_Prompt_Injection_Along_The_CIA_Security_Triad

[33] [41]: https://papers.nips.cc/paper_files/paper/2024/file/eb113910e9c3f6242541c1652e30dfd6-Paper-Confe
pdf

[34] [42]: https://feeds.transistor.fm/secure-talk-podcast

[35] [28]: https://www.krasamo.com/llm-security/

[36] [33]: https://www.reddit.com/r/LocalLLaMA/comments/1k0iu5z/announcing_realharm_a_collection_of_realworld/

[37] [36]: https://cetas.turing.ac.uk/publications/evaluating-malicious-generative-ai-capabilities

[38] [30]: https://eprints.whiterose.ac.uk/id/eprint/223407/1/Risk_of_What.pdf

[39] [35]: https://github.com/mlip-cmu/s2025/blob/main/lectures/23_safety/safety.md

[40] [31]: https://philarchive.org/archive/FEAROW

[41] [12]: https://www.cs.umd.edu/~jbg/docs/2023_emnlp_hackaprompt.pdf

[42] [44]: https://learnprompting.org/blog/hackaprompt-1-results

[43] [23]: https://www.procheckup.com/blogs/posts/2024/march/invisible-prompt-injection/

[44] [15]: https://iwcon.live/wp-content/uploads/2023/11/Main-Prompt-Injection-Implications-Attacks-and-Mi
compressed.pdf

[45] [38]: https://www.promptfoo.dev/blog/prompt-injection/

[46] [1]: https://www.researchgate.net/publication/386577011_Trust_No_AI_Prompt_Injection_Along_The_CIA_Security_Triad

[47] [69]: https://learnprompting.org/blog/what-is-model-context-protocol

[48] [70]: https://buttondown.com/ainews/archive/ainews-grok-2-and-chatgpt-4o-latest-confuses/

[49] [71]: https://simonwillison.net/tags/security/?page=2

[50] [37]: https://plextrac.com/the-new-artificial-intelligence/

[51] [72]: https://www.appsecpnw.org/speaker/jason_haddix/

[52] [47]: https://maven.com/learn-prompting-company/ai-red-teaming-and-ai-safety-masterclass

[53] [48]: https://paper.hackaprompt.com/

[54] [10]: https://arxiv.org/html/2501.11183v1

[55] [56]: https://www.researchgate.net/publication/388232623_Can_Safety_Fine-Tuning_Be_More_Principled_Lessons_Learned_from_Cybersecurity

[56] [73]: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-2e2025.pdf

[57] [18]: https://arxiv.org/html/2504.02080v1

[58] [19]: https://www.arxiv.org/pdf/2504.02080

[59] [14]: https://arxiv.org/html/2502.00735

[60] [25]: https://wtit.com/blog/2025/04/17/owasp-top-10-for-llm-applications-2025/

[61] [45]: https://coralogix.com/ai-blog/prompt-injection-attacks-in-llms-what-are-they-and-how-to-preven

[62] [39]: http://arxiv.org/pdf/2403.04786

[63] [53]: https://arxiv.org/html/2504.18333v1

[64] [54]: https://arxiv.org/html/2502.12630v1

[65] [55]: https://arxiv.org/html/2502.02260v1

[66] [32]: https://arxiv.org/pdf/2503.09780

[67] [74]: https://openreview.net/forum?id=fwHVclv0ij

[68] [43]: https://www.researchgate.net/publication/389821358_AgentDAM_Privacy_Leakage_Evaluation_for_Autonomous_Web_Agents

[69] [75]: https://e-discoveryteam.com/tag/defcon/

[70] [29]: https://www.deezer.com/us/show/4510667

[71] [76]: https://www.bugcrowd.com/blog/a-low-cost-hacking-sidekick-baby-steps-to-using-offensive-ai-age

[72] [77]: https://www.blackhat.com/us-24/sponsored-sessions/schedule/index.html

[73] [78]: https://www.blackhat.com/us-24/briefings/schedule/

[74] [79]: https://www.blackhat.com/us-25/ai-summit.html

[75] [20]: https://arxiv.org/html/2502.00735v1

[76] [21]: https://arxiv.org/pdf/2401.09002

[77] [67]: https://arxiv.org/html/2306.05499v2

[78] [52]: https://cloudnativenow.com/topics/containers/securing-genai-workloads-protecting-the-future-of-

[79] [50]: https://muckrack.com/john-mello/articles

[80] [26]: https://www.hackerone.com/blog/hackerone-and-owasp-top-10-llm-powerful-alliance-secure-ai