A Lifecycle and Application-Stack Survey of Large Language Model Vulnerabilities: Attacks, Risks, Defenses, and Open Problems

Seyed Bagher Hashemi Natanzi, Bo Tang
Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, Worcester, MA, USA
{snatanzi,btang1}@wpi.edu

Abstract

Large language models are no longer only text generators. They are increasingly embedded in retrieval pipelines, enterprise assistants, coding environments, robotic systems, security-operation workflows, and autonomous agents that can read private data, call tools, write files, execute code, and act across organizational boundaries. This shift changes the security problem: risks do not arise from the model weights alone, but from the full lifecycle and application stack through which data, prompts, model outputs, tools, memories, and user authority interact. This paper systematizes the literature on vulnerabilities in large language model systems through a lifecycle and application-stack lens. We organize attacks across eight stages: data collection, pretraining, post-training alignment, model packaging and supply chain, retrieval and memory, prompting and inference, tool/agent execution, and deployment/maintenance. For each stage, we analyze attacker capabilities, affected security objectives, representative attacks, practical risks, evaluation practices, and defenses. We further map LLM-specific vulnerabilities to confidentiality, integrity, availability, safety, privacy, fairness, accountability, and agency-control objectives. Unlike taxonomies that list isolated attack names, the proposed systematization emphasizes where trust boundaries fail, how untrusted data becomes executable instruction, how delegated authority amplifies model errors, and why point defenses rarely compose. We close with a research agenda for secure LLM systems, including compositional security, provenance-aware retrieval, tool-call containment, long-horizon agent evaluation, privacy-preserving adaptation, realistic red teaming, and deployment-grade incident response.

Keywords: large language models, LLM security, prompt injection, jailbreak, retrieval-augmented generation, LLM agents, data poisoning, privacy, systematization of knowledge, trustworthy AI

1 Introduction

Transformer-based language models have become a general-purpose substrate for natural-language interfaces, information retrieval, code generation, planning, and multimodal reasoning (Brown et al., 2020; Devlin et al., 2019; Gemini Team, 2023; OpenAI, 2023; Vaswani et al., 2017). Their adoption has also shifted from model-centric research demonstrations to application stacks that combine model APIs, retrieval systems, user profiles, persistent memories, external tools, plug-ins, workflow orchestrators, and autonomous agents (Lewis et al., 2020; Qin et al., 2023; Schick et al., 2023; Yao et al., 2023). This deployment shift changes the nature of security. A model that only emits text may cause unsafe content or misinformation; an application that lets the same model access files, email, calendars, databases, cloud APIs, browsers, terminals, or robots can convert text-generation failures into external side effects.

The resulting risk landscape is broad. Training data can contain private information, copyrighted content, low-quality sources, malicious triggers, or biased patterns (Bender et al., 2021; Bommasani et al., 2021; Weidinger et al., 2021). Pretrained models can memorize rare sequences, reveal training examples, or expose privacy information through extraction and inference attacks (Carlini et al., 2019, 2021; Shokri et al., 2017). Alignment and instruction tuning can reduce many harms but can also be attacked, bypassed, or weakened by fine-tuning (Bai et al., 2022; Ouyang et al., 2022; Qi et al., 2023; Rafailov et al., 2023). Prompting interfaces are vulnerable to adversarial suffixes, jailbreaks, prompt leakage, direct prompt injection, and indirect prompt injection through untrusted external content (Greshake et al., 2023; Liu et al., 2023a; Perez and Ribeiro, 2022; Wei et al., 2023; Zou et al., 2023). Retrieval-augmented generation (RAG) introduces document ingestion, embedding, vector search, provenance, and stale-memory risks (Karpukhin et al., 2020; Lewis et al., 2020; Xiong et al., 2024; Zhao et al., 2024). Tool-using agents add delegated authority, state, persistent memory, multi-step planning, and protocol-level attack surfaces (Chu, 2026; Ferrag et al., 2025; Ling et al., 2026; Yao et al., 2023; Zhao et al., 2026).

Existing surveys have made important progress, but the field remains fragmented. Some surveys organize risks by attack technique, such as prompt injection, poisoning, jailbreaking, or privacy leakage (Xu et al., 2025; Yao et al., 2024). Others focus on LLMs in general, alignment, trustworthiness, or agent-specific security (Chu, 2026; Derner and Batistic, 2024; Ling et al., 2026; Liu et al., 2023b; Naveed et al., 2023). Industry frameworks such as OWASP's LLM Top 10 and NIST's adversarial machine-learning taxonomy provide operational vocabulary and mitigation categories (OWASP GenAI Security

Project, 2025; Vassilev et al., 2025). However, developers and researchers still need a unifying systems view that answers not only *what* the attack is called, but *where* it arises in the lifecycle, *which* trust boundary failed, *what* authority the model had, and *how* defenses should compose across the stack.

We argue that LLM vulnerabilities are best understood as failures of information flow, provenance, privilege, and state across a lifecycle and application stack. The same surface string can be harmless in a sandboxed chatbot but dangerous in an agent with database write privileges. Similarly, the same poisoning attempt has different implications when it appears in pretraining data, an instruction-tuning set, a RAG corpus, a tool description, or a persistent memory. A systematization-of-knowledge survey should therefore connect attacks to lifecycle stage, security objective, attacker capability, and defense layer. In summary, this paper makes four contributions.

1. We propose a lifecycle and application-stack taxonomy for LLM vulnerabilities, covering data collection, pretraining, post-training alignment, model packaging and supply chain, RAG/memory, prompting and inference, tool/agent execution, and deployment/maintenance.

2. We map attacks to security objectives beyond the traditional confidentiality–integrity–availability triad, adding safety, privacy, fairness, accountability, and agency control as first-class objectives for LLM applications.

3. We synthesize defenses into a defense-in-depth architecture that separates deterministic controls, model-level robustness, monitoring/red teaming, privacy controls, and governance processes.

4. We identify open problems that remain under-specified in current literature, including compositional prompt-injection defenses, provenance-preserving RAG, secure tool delegation, stateful/multi-agent evaluation, supply-chain integrity for model artifacts, and incident response for adaptive LLM systems.

2 Scope, Method, and Terminology

2.1 Scope

We use *large language model system* to refer to a deployed system that includes one or more language or multimodal foundation models, prompts, safety policies, retrieval components, context windows, memory stores, tool-call interfaces, plug-ins, monitoring components, and human operators. This scope is broader than the base model. It includes model weights, but also the application stack in which the model is embedded.

We focus on vulnerabilities that are specific to, amplified by, or operationally transformed by LLMs. General web, cloud, identity, and software-security vulnerabilities are included only when LLM integration changes the attack path or impact. For instance, SQL injection is a classical vulnerability; it becomes part of this survey when an LLM generates SQL from natural language or when untrusted database content becomes part of an LLM prompt. We also include multimodal and agentic systems when the LLM acts as the central planner or interface.

2.2 Review protocol and coding schema

The goal is systematization rather than exhaustive bibliometrics. We used seed papers from LLM security, privacy, adversarial ML, RAG, and agent-security literature; expanded by citation chasing; and cross-checked against practitioner taxonomies including OWASP LLM Top 10, MITRE ATLAS, and NIST AI 100-2e2025 (MITRE, 2024; OWASP GenAI Security Project, 2025; Vassilev et al., 2025). We coded representative papers along the following fields:

- **Lifecycle stage:** data, pretraining, alignment, packaging/supply chain, RAG/memory, prompt/inference, tool/agent execution, deployment/maintenance.

- **Security objective:** confidentiality, integrity, availability, safety, privacy, fairness, accountability, agency control.

- **Attacker capability:** black-box prompting, query access, white-box access, data-injection access, tool-description control, RAG-corpus write access, fine-tuning access, supply-chain access, insider access, or deployment access.

- **Attack mechanism:** injection, evasion, jailbreak, extraction, poisoning, backdoor, inversion, membership inference, tool misuse, retrieval manipulation, prompt leakage, or resource exhaustion.

- **Defense layer:** data governance, model training, prompt/context management, retrieval controls, privilege isolation, output verification, monitoring, red teaming, privacy engineering, supply-chain assurance, or incident response.

This protocol intentionally avoids treating attack names as mutually exclusive categories. Many real attacks are compositional. For example, an indirect prompt injection can be delivered through a poisoned RAG document, hidden with Unicode obfuscation, cause prompt leakage, and invoke an over-privileged tool. A lifecycle coding makes such composition explicit.

2.3 Terminology

Vulnerability denotes a weakness in the design, implementation, integration, or operation of an LLM system that can be exploited or accidentally triggered. **Threat** denotes a potential harmful event enabled by a vulnerability. **Attack** denotes an intentional action by an adversary. **Risk** combines likelihood, exposure, and impact. **Defense** denotes a preventive, detective, corrective, or compensating control.

We distinguish **jailbreaking** from **prompt injection**. Jailbreaking usually refers to inducing a model to violate safety policies. Prompt injection refers to causing the model or application to treat attacker-controlled content as instructions. Jailbreaking can be a subclass or goal of prompt injection, but

Table 1: Attacker models for lifecycle and application-stack analysis.

Attacker model	Access and knowledge	Typical attacks	Relevant defenses
Black-box prompt user	Can send prompts and observe outputs; may know public model family but not hidden prompts or weights	jailbreaks, prompt leakage, social-engineering prompts, resource exhaustion	refusal training, rate limits, output filtering, abuse monitoring, prompt-injection detectors
Adaptive query adversary	Can submit many queries and optimize based on responses	adversarial suffix search, model extraction, privacy probing, transfer attacks	query throttling, anomaly detection, randomized defenses, privacy auditing
External-content adversary	Controls content that may later be retrieved or summarized, such as webpages, PDFs, emails, tickets, or repositories	indirect prompt injection, RAG poisoning, malicious tool observations	provenance labels, source allowlists, context isolation, retrieval filtering, least privilege
Corpus/data adversary	Can influence pretraining, instruction-tuning, preference, or evaluation data	data poisoning, backdoors, benchmark contamination, alignment poisoning	data provenance, deduplication, dataset signing, poisoning scans, held-out private evaluations
Fine-tuning/adaptation adversary	Can provide adapters, fine-tuning data, or model updates	safety degradation, hidden triggers, policy drift	safety regression tests, adapter scanning, signed updates, restricted fine-tuning APIs
Supply-chain adversary	Can tamper with model artifacts, tokenizers, packages, plug-ins, or tool manifests	artifact compromise, malicious loaders, compromised plug-ins, tokenizer attacks	signed artifacts, sandboxing, SBOM/MBOM, dependency scanning, reproducible builds
Insider or operator	Has privileged access to logs, prompts, data stores, model versions, or deployment settings	data exfiltration, unsafe policy changes, logging abuse, rollback prevention	separation of duties, audit logs, access control, approval workflows, key management

prompt injection also includes data exfiltration, tool misuse, goal hijacking, and prompt leakage (Greshake et al., 2023; Liu et al., 2023a; Perez and Ribeiro, 2022). We use **indirect prompt injection** for attacks where malicious instructions are embedded in external content, such as a document, webpage, email, code repository, tool result, or retrieval item, and later enter the prompt through application logic.

2.4 Attacker models

A useful taxonomy must separate the attack surface from the attacker's capability. We use the attacker models in Table 1. These models are not mutually exclusive; a realistic adversary may start as a black-box user, poison a public webpage, wait for it to be retrieved, and then use the resulting agent behavior to obtain additional access.

3 Lifecycle and Application-Stack Taxonomy

Figure 1 illustrates our organizing view. The lifecycle stages are not merely chronological; they are also security boundaries. Data collection determines what the model may memorize or normalize. Pretraining compresses public and private corpora into parameters. Post-training alignment changes refusal behavior and instruction following. Packaging and distribution determine whether users can trust weights, adapters, tokenizer files, and inference code. RAG and memory create mutable knowledge surfaces. Prompting and inference expose the model to direct adversarial interaction. Tool and agent execution delegates authority. Deployment and maintenance determine monitoring, rollback, patching, and incident response.

Table 2 shows why a lifecycle view matters. A single term such as "poisoning" is insufficient because the object being poisoned changes: pretraining data, alignment data, RAG corpora, tool descriptions, memory, evaluation benchmarks, or monitoring logs. Likewise, a single term such as "prompt injection" is insufficient because the payload can be supplied directly by the user, indirectly by retrieved content, persistently through stored memory, or transitively through tool outputs.

3.1 Security objectives

Traditional security uses confidentiality, integrity, and availability. LLM systems require these objectives but also extend them.

- **Confidentiality:** prevent unauthorized disclosure of user data, system prompts, proprietary documents, credentials, hidden chain instructions, and training data.
- **Integrity:** preserve correctness of model behavior, retrieved context, tool arguments, outputs, logs, and persistent memories.
- **Availability:** maintain service quality under prompt floods, long-context resource exhaustion, expensive tool loops, or denial-of-wallet attacks.
- **Safety:** prevent harmful instructions, dangerous recommendations, policy bypass, and unsafe execution.

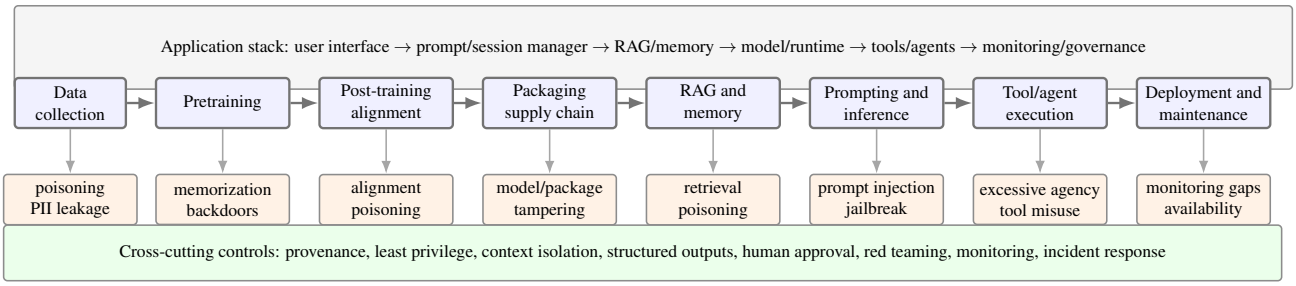

Figure 1: Lifecycle and application-stack view of LLM vulnerabilities. The same attack technique can appear at multiple stages, but its feasibility, impact, and defensibility depend on where trust boundaries, data flows, and delegated authority are located.

- **Privacy:** reduce memorization, membership inference, inference about sensitive attributes, and cross-user leakage.
- **Fairness:** avoid disproportionate errors, toxic outputs, stereotyping, and exclusionary service behavior.
- **Accountability:** support auditability, provenance, explainability of actions, rollback, and responsibility assignment.
- **Agency control:** ensure the model cannot exceed user intent, tool privileges, or organizational policy when acting in the world.

4 Attack-Family Synthesis

The lifecycle taxonomy answers where a vulnerability appears. This section synthesizes the major attack families across stages. The goal is to connect mechanisms that are often studied separately. Table 3 maps the LLM attack families to security objectives and common attacker capabilities, and Table 4 summarizes attack-family synthesis across lifecycle stages.

4.1 Privacy leakage, extraction, and inference

Privacy attacks exploit the fact that LLMs are trained on, adapted with, or connected to sensitive information. Training-data extraction recovers memorized text from model completions, especially when sequences are rare, duplicated, or over-represented (Carlini et al., 2021). Membership inference estimates whether a candidate sample participated in training (Nasr et al., 2019; Shokri et al., 2017). Model inversion attempts to reconstruct representative inputs or sensitive attributes (Fredrikson et al., 2015). In RAG systems, privacy leakage may occur even if the base model is safe: a user may retrieve documents they should not access, an embedding may reveal information through nearest-neighbor behavior, or a prompt injection may cause an agent to summarize private context.

Privacy evaluation must therefore specify the data location. Leakage from weights, context windows, retrieval stores, logs, telemetry, memories, and tool outputs require different mitigations. A memorization defense will not protect against over-permissive retrieval. A retrieval access-control policy will not protect against secrets placed directly inside system prompts. This is why privacy should be evaluated as an end-to-end property of the application stack.

4.2 Adversarial prompting, jailbreaks, and evasion

Adversarial prompting includes direct prompt injection, jailbreaks, adversarial suffixes, role-play attacks, multilingual attacks, encoding/obfuscation, and multi-turn persuasion (Perez and Ribeiro, 2022; Wallace et al., 2019; Wei et al., 2023; Zou et al., 2023). These attacks exploit the model's instruction-following objective and the ambiguity of natural language. The most important distinction is between attacks that merely elicit unsafe text and attacks that alter application control flow. In a tool-free chatbot, a jailbreak may produce policy-violating content. In a connected agent, the same jailbreak can become a command to read, write, send, or execute.

Evasion also targets safety classifiers and guard models. Attackers may change formatting, scripts, Unicode characters, word boundaries, or language to bypass filters while preserving meaning. Such attacks demonstrate that lexical filtering alone is fragile. Robust systems combine normalization, semantic detection, model-level robustness, and post-generation action controls.

4.3 Prompt leakage and system-prompt exposure

Prompt leakage attempts to reveal hidden instructions, policies, examples, tool descriptions, or developer prompts. The security impact depends on what the prompt contains. If the prompt contains only general behavioral guidance, leakage may be low impact. If it contains credentials, private data, proprietary workflows, or security policies that enable bypass, leakage becomes a confidentiality and integrity risk. Good design minimizes secrets in prompts and treats prompts as potentially recoverable configuration, not secure storage.

4.4 Poisoning, backdoors, and alignment attacks

Poisoning alters data or feedback so that the model learns attacker-preferred behavior. Backdoors are a special case in

Table 2: Lifecycle-stage view of LLM vulnerabilities, representative risks, and defense families. The table is intended as a systems checklist rather than an exhaustive paper catalogue.

Stage	Typical vulnerabilities	Representative risks	Defense families
Data collection and curation	PII in corpora, biased sources, low-quality or duplicated data, malicious documents, weak provenance	memorization, privacy leakage, data poisoning, representational harms	data documentation, deduplication, PII filtering, provenance tracking, dataset access control, poisoning scans
Pretraining	memorization of rare strings, backdoor triggers, insufficient data isolation, weak privacy accounting	training-data extraction, latent backdoors, unauthorized knowledge retention	privacy-aware training, deduplication, canary tests, backdoor detection, secure training infrastructure
Post-training alignment	poisoned instruction data, unsafe preference data, reward hacking, fine-tuning that removes safeguards	alignment degradation, policy bypass, hidden backdoors, over-refusal or under-refusal	dataset auditing, red-team SFT/RLHF/DPO, alignment regression tests, safety-preserving fine-tuning
Packaging and supply chain	tampered weights, malicious adapters, unsafe model code, tokenizer manipulation, vulnerable dependencies	model compromise, remote code execution, hidden triggers, supply-chain attacks	signed artifacts, reproducible builds, sandboxed loaders, dependency scanning, model cards, software bills of materials
RAG and memory	poisoned retrieval corpora, stale documents, prompt-like content in retrieved data, embedding leakage, cross-user memory contamination	indirect prompt injection, retrieval hijacking, sensitive disclosure, misinformation	source allowlists, provenance labels, retrieval-time filtering, context separation, memory isolation, freshness checks
Prompting and inference	direct injection, jailbreak, adversarial suffixes, prompt leakage, side-channel prompting, output hallucination	safety-policy bypass, developer-prompt disclosure, data exfiltration, harmful content	prompt hardening, input normalization, instruction hierarchy, output verification, refusal calibration, rate limits
Tool and agent execution	excessive privileges, untrusted tool outputs, weak schemas, cross-tool confused deputy, persistent state corruption	unauthorized action, file/database modification, credential leakage, multi-agent propagation	least privilege, capability scoping, structured tool APIs, human approval, sandboxing, transaction logs
Deployment and maintenance	weak monitoring, insecure updates, exposed endpoints, no rollback, resource exhaustion, misconfigured logging	availability loss, privacy breach, delayed detection, incident amplification	observability, anomaly detection, access control, patch management, canary deployment, incident response

which the model behaves normally except when triggered (Chen et al., 2017; Gu et al., 2017; Kurita et al., 2020). In LLMs, poisoning can occur at pretraining, instruction tuning, preference optimization, RAG ingestion, memory updates, tool descriptions, or benchmark construction. Alignment attacks target the post-training stage by poisoning safety examples or weakening refusal behavior during fine-tuning (Qi et al., 2023; Wan et al., 2023). The clean-performance constraint makes these attacks difficult to detect: the model may pass standard benchmarks while failing on trigger contexts.

Defenses require provenance and regression testing. Data should be traceable; safety tests should include trigger-like contexts; adapters should be scanned before merging; and fine-tuned models should be compared against their base models on safety, privacy, and utility. Benchmark contamination must also be monitored because LLMs may memorize public evaluation sets, leading to overestimated robustness.

4.5 RAG, vector databases, and memory attacks

RAG attacks manipulate what the model sees at inference time. Retrieval poisoning inserts documents that are likely to be retrieved for target queries. Indirect prompt injection embeds instructions inside external content. Vector-store attacks may exploit embedding similarity, chunk boundaries, stale indexes, or missing access control (Xiong et al., 2024; Zhao et al., 2024). Memory attacks persist false or malicious state across sessions.

The key insight is that RAG turns content security into instruction security. A document is no longer merely read by a human; it is parsed by a model that may treat it as a command. The defense is not to ban RAG, but to preserve source trust, label untrusted content, and prevent retrieved text from authorizing actions.

4.6 Tool, plug-in, and agent attacks

Agents amplify LLM failures because they connect model outputs to external effects. Tool attacks include malicious tool outputs, prompt injection through API responses, unsafe command construction, tool-description poisoning, excessive permissions,

Table 3: Mapping LLM attack families to security objectives and common attacker capabilities. A filled cell indicates a frequent primary impact; many attacks have secondary impacts.

Attack family	Common capability	Conf.	Integ.	Avail.	Safety	Privacy	Fairness	Account.	Agency
Training-data extraction	query access; memorization probes	•				•			
Membership inference	query or confidence access	•				•			
Model extraction	API access, distillation budget	•	•						
Poisoning/backdoors	data or fine-tuning access		•		•		•		
Jailbreak/adversarial suffix	black-box prompt access		•		•				
Direct prompt injection	user input control	•	•		•	•			•
Indirect prompt injection	external content control	•	•		•	•		•	•
RAG poisoning	corpus or source control	•	•		•	•		•	
Tool misuse/excessive agency	prompt or tool-output control	•	•	•	•	•		•	•
Resource exhaustion	query and context budget			•					
Supply-chain compromise	artifact/dependency control	•	•	•	•	•		•	•

and cross-tool data exfiltration (Ferrag et al., 2025; Ling et al., 2026; Zhao et al., 2026). Multi-agent systems add propagation risks: one compromised agent can influence another through messages, shared memory, or delegated tasks.

The most dangerous pattern is the confused deputy. The user delegates authority to the agent; the agent reads attacker-controlled content; the content persuades the agent to use the user's authority for the attacker's goal. Preventing this requires explicit authority tracking. The system should know which principal authorized each action and which untrusted content influenced it.

4.7 Availability, cost, and abuse

LLM availability attacks include prompt floods, long-context resource exhaustion, retrieval amplification, expensive tool loops, recursive agent calls, and denial-of-wallet attacks. Abuse also includes using LLMs to scale phishing, spam, misinformation, malware assistance, or vulnerability discovery. While these harms are not always vulnerabilities in the model itself, they are risks of deployment. Defenses include quotas, abuse detection, proof-of-work or payment controls for high-volume access, tool-call limits, circuit breakers, and misuse monitoring.

5 Threats Across the Lifecycle

5.1 Data collection and curation

Large-scale pretraining relies on web crawls, code repositories, books, forums, scientific articles, product documentation, synthetic data, and human-generated instruction examples. The main vulnerabilities are weak provenance, unknown consent, duplication, private information, toxic or biased content, and malicious data insertion. The data stage creates both model-level and system-level risks. A model may memorize rare strings or private records (Carlini et al., 2019, 2021); it may reproduce stereotypes or harmful associations (Bender et al., 2021; Weidinger et al., 2021); or it may learn trigger-response behavior if attackers insert poisoned documents before training (Chen et al., 2017; Gu et al., 2017; Kurita et al., 2020).

A lifecycle perspective distinguishes passive contamination from active poisoning. Passive contamination includes accidental inclusion of personal information, outdated facts, or biased sources. Active poisoning includes attacker-selected text that induces a future behavior, such as a backdoor trigger, malicious code pattern, or false association. LLM data scale does not eliminate this risk; scale can dilute triggers, but it also makes exhaustive manual auditing infeasible. Synthetic-data pipelines add another complication: if model-generated content re-enters training data without provenance, errors and attack artifacts can become self-reinforcing.

Defenses at this stage should be preventive. They include dataset documentation, source reputation scoring, near-duplicate removal, PII detection, toxicity screening, canary insertion for memorization tests, poisoning anomaly detection, and access control on data pipelines. These defenses have tradeoffs: aggressive filtering can remove minority dialects or specialized technical content, while weak filtering can preserve private or malicious data. The research gap is not only better filters, but auditable data governance that can answer which sources influenced which model behavior.

5.2 Pretraining and model internals

Pretraining compresses large corpora into parameters. This compression can create privacy leakage and hidden integrity

Table 4: Attack-family synthesis across lifecycle stages. The same family can appear at multiple layers with different defenses.

Family	Lifecycle manifestations	Primary failure mode	Most useful controls
Privacy and extraction	pretraining memorization, prompt leakage, RAG over-retrieval, log exposure, memory leakage	confidential information becomes accessible through generation or retrieval	privacy-aware training, access control, secret minimization, log redaction, memory isolation
Prompt injection and jailbreak	direct user prompts, indirect retrieved content, tool outputs, multimodal inputs, stored memories	untrusted language is interpreted as instruction	context isolation, instruction hierarchy, detectors, least privilege, output/action verification
Poisoning and backdoors	data collection, instruction tuning, preference data, RAG corpora, adapters, tool manifests	attacker changes future behavior while preserving normal utility	provenance, data signing, anomaly scans, trigger tests, safety regression suites
Supply-chain compromise	model files, adapters, tokenizers, inference code, dependencies, plug-ins	trusted artifact contains malicious behavior or vulnerable code	signed artifacts, sandboxed loading, dependency scanning, MBOM/SBOM, reproducible builds
Agent/tool misuse	tool schemas, API calls, code execution, email/browser/file access, multi-agent messages	model output causes unauthorized external effect	capability scoping, human approval, dry-run mode, transaction logs, policy engines
Availability and cost abuse	long context, repeated queries, retrieval/tool loops, expensive generation, API amplification	attacker consumes resources or degrades service	rate limits, quotas, timeouts, circuit breakers, billing anomaly detection

failures. Extraction attacks exploit model completions to recover memorized training sequences, especially rare or duplicated strings (Carlini et al., 2021). Membership inference attacks attempt to determine whether a sample was present in training (Nasr et al., 2019; Shokri et al., 2017). Model inversion aims to infer properties or representative inputs from outputs or internal representations (Fredrikson et al., 2015). Model extraction and distillation attacks aim to replicate functionality through API queries (Tramèr et al., 2016).

Backdoors and weight poisoning are integrity threats. In a backdoored model, behavior appears normal except when a trigger activates malicious behavior (Gu et al., 2017; Kurita et al., 2020). For LLMs, triggers can be lexical, syntactic, semantic, multilingual, or embedded in code comments. Unlike image classifiers, LLMs may execute backdoor behavior through long-form generation or tool calls, so evaluation must measure not only class accuracy but downstream action.

Mitigations include deduplication, privacy-aware training, differential privacy, controlled memorization tests, gradient clipping, red-team prompts, backdoor scanning, and model-card disclosure (Abadi et al., 2016; Carlini et al., 2019; Dwork et al., 2006). Differential privacy can reduce memorization but may degrade utility at large model scale if not carefully engineered. Backdoor detection remains difficult when triggers are semantic rather than token-level. A practical defense should combine data curation, training-time tests, inference-time monitoring, and limited authority for downstream actions.

5.3 Post-training alignment and fine-tuning

Instruction tuning, RLHF, RLAIF, constitutional AI, and DPO improve helpfulness and safety but also introduce new attack surfaces (Bai et al., 2022; Ouyang et al., 2022; Rafailov et al., 2023; Stiennon et al., 2020; Ziegler et al., 2019). Alignment datasets can be poisoned, preference models can encode hidden policies, and downstream fine-tuning can weaken refusal behavior (Qi et al., 2023; Wan et al., 2023). The alignment stage is therefore both a defense and a target.

Post-training vulnerabilities have three common forms. First, poisoned examples can teach a model to comply with unsafe requests under specific contexts. Second, preference optimization can overfit to superficial refusal templates, making the model brittle under rephrasing, role play, or multi-turn pressure. Third, benign fine-tuning for domain adaptation can erode safety boundaries because the optimization objective rewards task compliance rather than safety preservation.

Defenses should treat safety as a regression property. Any fine-tuning or adapter update should be tested against a stable suite of safety, privacy, jailbreak, and task-utility prompts. Safety-preserving fine-tuning should freeze or regularize parts of the model when possible, include refusal and boundary examples, and evaluate cross-domain transfer. The field still lacks reliable certificates that an aligned model remains aligned after adapter merging, quantization, distillation, or local fine-tuning.

5.4 Packaging, distribution, and supply chain

Open models, adapters, tokenizers, inference servers, model-loading scripts, and third-party packages create a software supply chain. LLM systems frequently load artifacts from public model hubs, run custom Python code, and combine multiple dependencies. Vulnerabilities include malicious model files, unsafe deserialization, compromised adapters, tokenizer manipulation, dependency confusion, prompt templates hidden in packages, and contaminated evaluation scripts.

The risk is amplified because model artifacts are large,

opaque, and hard to inspect. A malicious adapter can change behavior without modifying base weights. A tokenizer can alter how dangerous strings are segmented. A model repository can include code that executes at load time. Supply-chain attacks can also occur through plug-ins, tool manifests, browser extensions, MCP-like servers, and agent skills (Chu, 2026; Dehghantanha et al., 2026; Ferrag et al., 2025).

Defenses should import mature software-security practices into ML operations: signed artifacts, hashes, reproducible builds, sandboxed model loading, dependency scanning, software bills of materials, model bills of materials, vulnerability disclosure, least-privilege inference containers, and provenance labels. Model cards should be extended to security cards that disclose training data classes, alignment methods, tool privileges, known limitations, and intended deployment boundaries.

5.5 Retrieval-augmented generation and memory

RAG connects an LLM to external knowledge through retrieval and context construction (Karpukhin et al., 2020; Lewis et al., 2020). It reduces some hallucinations and supports private-domain applications, but it introduces a new pathway by which untrusted data becomes model context. A retrieved document can contain instructions that override the user's goal, leak system prompts, exfiltrate information, or steer summaries. This is the canonical setting for indirect prompt injection (Greshake et al., 2023; Liu et al., 2023a). Retrieval can also be poisoned by adding documents that dominate nearest-neighbor search, manipulate embeddings, or appear authoritative (Xiong et al., 2024; Zhao et al., 2024).

RAG creates three distinct trust problems. First, retrieved content has uncertain provenance. Second, retrieved content is mixed with developer instructions inside the model context. Third, retrieved content is often treated as evidence even when it is stale, adversarial, or out of distribution. Persistent memory adds a temporal dimension: a malicious or mistaken memory can affect future sessions, and cross-user memory bugs can become privacy breaches.

Defenses include source allowlists, retrieval-time safety classification, provenance metadata, freshness checks, user-visible citations, context partitioning, and memory isolation. The model should not be asked to infer which text is instruction and which text is data solely from natural-language formatting. Application logic should label untrusted content, limit what retrieved content can authorize, and verify claims against multiple sources when high impact. Research is needed on retrieval systems that preserve provenance through embeddings and context construction rather than losing it at the vector-search boundary.

5.6 Prompting and inference

Prompting is the most visible attack surface. Direct prompt injection attempts to hijack goals, reveal hidden prompts, bypass policies, or override previous instructions (Perez and Ribeiro, 2022). Jailbreaks and adversarial suffixes exploit instruction-following behavior and transfer across models (Wei et al., 2023; Zou et al., 2023). Universal adversarial triggers show that small textual patterns can systematically change model behavior (Wallace et al., 2019). Programmatic behavior can also be exploited when models process code-like or instruction-like structures (Kang et al., 2023).

The root cause is not simply that models are insufficiently aligned. The root cause is that natural language is used simultaneously as data, instruction, policy, evidence, and executable plan. Delimiters, system prompts, and refusal templates help but do not create cryptographic separation. Models can be confused by role-play, translation, encoding, multi-turn pressure, conflicting instructions, hidden Unicode, or tool outputs. Larger context windows increase the volume of untrusted content that may affect behavior.

Defenses include input normalization, Unicode and encoding governance, prompt templates that separate roles, instruction hierarchy, prompt-injection detectors, output safety filters, rate limiting, and robust refusal training. However, these defenses are probabilistic when they rely on the model to classify intent. Deterministic controls should therefore constrain what the model can do after generation. For example, the model may propose a database query, but a schema validator and policy engine should decide whether the query can run.

5.7 Tools, plug-ins, and agents

Tool-using LLMs convert language into actions. ReAct-style prompting, Toolformer-like tool use, API-augmented models, and agent frameworks allow models to plan, retrieve, compute, call APIs, and update state (Qin et al., 2023; Schick et al., 2023; Yao et al., 2023). This creates new risks: excessive agency, tool-output prompt injection, confused-deputy behavior, insecure tool schemas, credential leakage, multi-step escalation, and persistent state corruption. In security operations, LLMs may help triage vulnerabilities or automate penetration-testing tasks, but this also increases dual-use concerns (Deng et al., 2023).

Agent security differs from chatbot security in three ways. First, agents possess delegated authority. A harmful output may become an email, transaction, code commit, database update, or shell command. Second, agents are stateful. They store memories, partial plans, and tool observations across steps. Third, agents interact with untrusted environments. Web pages, repositories, tickets, emails, and PDFs can contain instructions targeting the agent rather than the human user. Recent work therefore treats LLM-agent ecosystems as protocol and workflow security problems, not only model-safety problems (Chu, 2026; Dehghantanha et al., 2026; Ferrag et al., 2025; Ling et al., 2026; Zhao et al., 2026).

Defenses must control authority explicitly. Tools should be least-privilege, scoped to the user's current task, and separated by sensitivity. Tool calls should use typed schemas, not free-form strings. High-impact operations should require human approval. Tool outputs should be treated as untrusted data and routed through the same context-isolation and normalization pipeline as web content. Sandboxing, dry-run modes, transaction logs, and reversible execution reduce impact. A central research challenge is compositionality: a system can have safe

components but unsafe interactions when a model chains them.

5.8 Deployment, monitoring, and maintenance

Deployment determines whether failures are detected, contained, and corrected. LLM applications can fail through exposed endpoints, weak authentication, logs containing secrets, excessive context costs, denial-of-wallet attacks, prompt floods, tool loops, unbounded recursion, model-version drift, stale safety prompts, and missing incident response. Maintenance adds update channels, rollback procedures, dependency patching, prompt-template changes, and red-team regression tests.

Availability is particularly under-discussed. Long-context models can be expensive; attackers may send inputs that maximize token usage, retrieval load, or tool calls. Agents can enter loops or repeatedly call costly APIs. Defensive design should include token budgets, tool-call quotas, timeouts, circuit breakers, queue isolation, and billing anomaly detection. Monitoring should not only log prompts and completions; it should log provenance, retrieved document IDs, tool arguments, policy decisions, user approvals, model version, and safety-detector outputs.

Incident response for LLM systems differs from conventional software response. A fix may involve changing prompts, safety policies, retrieval filters, memory entries, model versions, adapters, tool permissions, or data sources. Some incidents require forgetting or quarantining memories; others require retraining or revoking model artifacts. Organizations therefore need LLM-specific runbooks for prompt-injection incidents, data leakage, unsafe tool use, poisoning discovery, and model rollback.

6 Defense-in-Depth Architecture

No single defense is sufficient. Model alignment reduces risk but cannot guarantee that the model will correctly separate data from instructions in all contexts. Prompt filtering catches some attacks but can be bypassed or over-block benign use. RAG citations improve transparency but do not prevent malicious retrieved content from influencing behavior. Tool schemas reduce injection into APIs but not necessarily goal hijacking. Therefore, the core design principle is defense in depth: combine deterministic controls that bound actions with probabilistic controls that detect and reduce residual risk. Figure 2 summarizes a general defense-in-depth architecture for LLM applications.

6.1 Deterministic controls

Deterministic controls should be preferred wherever a security property can be expressed outside the model. Examples include least-privilege tool permissions, allowlisted network destinations, typed API schemas, JSON schema validation, sandboxed code execution, read-only defaults, transaction limits, human approval for irreversible operations, retrieval-source allowlists, context-size budgets, and cryptographic signing of model artifacts. These controls are auditable and do not require the model to reason perfectly under adversarial input.

A useful rule is that untrusted natural language should not directly authorize external actions. It may provide evidence, a candidate plan, or a draft, but application logic should verify whether the requested action is within scope. For example, a retrieved document may say that an agent should send a file to a URL, but the system should treat that instruction as data and reject it unless the authenticated user independently authorized the action.

6.2 Model-level and prompt-level controls

Model-level defenses include adversarial training, refusal calibration, safety fine-tuning, preference optimization, constitutional rules, and robustness evaluation (Bai et al., 2022; Ganguli et al., 2022; Ouyang et al., 2022; Rafailov et al., 2023). Prompt-level defenses include system prompts, instruction hierarchy, delimiters, reminders about untrusted content, self-check prompts, and chain-of-verification. These controls improve behavior but remain statistical. They should be evaluated continuously against adaptive attacks and regressions.

Prompt hardening is most useful when paired with architectural separation. A prompt may tell the model that retrieved content is untrusted; a system design should also ensure that retrieved content cannot directly set tool permissions, change policy, or erase logs. Similarly, a model may be trained to refuse prompt-leakage requests; the application should still avoid placing unnecessary secrets in prompts.

6.3 Retrieval and memory controls

RAG defenses should operate before, during, and after retrieval. Before retrieval, systems should curate sources, scan documents, attach provenance, and limit ingestion rights. During retrieval, systems should filter malicious-looking content, diversify sources, and preserve metadata. After retrieval, systems should label context, quote evidence separately from instructions, and verify high-impact claims. Persistent memory should be scoped by user, organization, task, and time; sensitive memory should expire or require explicit confirmation.

Embedding systems need additional controls. Embeddings can leak information, and nearest-neighbor search can be manipulated by adversarial documents. Access control should be enforced before retrieval rather than after the vector search whenever possible. If a user lacks access to a document, its embedding should not influence retrieval results for that user. Provenance should not be discarded when text is chunked and embedded.

6.4 Tool and agent controls

Tools should be designed as capability objects rather than broad ambient authority. Each tool should declare its scope, inputs, outputs, side effects, and risk level. Agents should receive only the tools needed for the current task. Sensitive tools should support dry-run, preview, confirmation, and rollback. Tool outputs should be untrusted observations, not instructions.

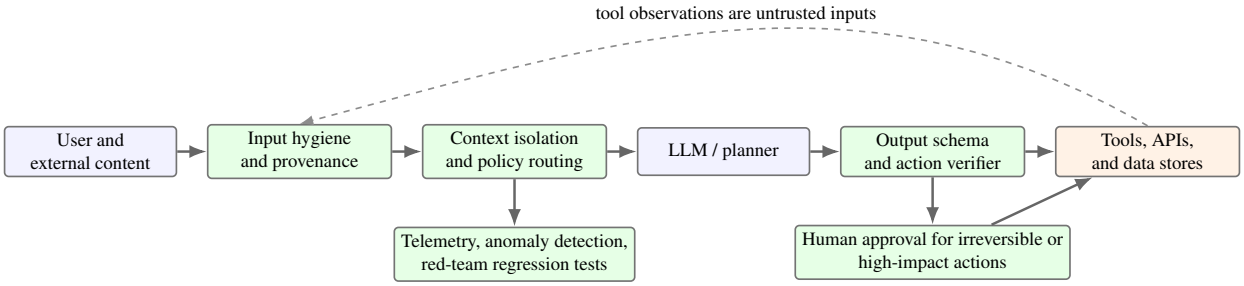

Figure 2: Defense-in-depth architecture for LLM applications. Reliable mitigation cannot rely only on model behavior; deterministic controls should constrain information flow and delegated authority before and after the model call.

Cross-tool data flow should be mediated by policy: a web-reading tool should not be able to instruct an email-sending tool to exfiltrate private data.

Agent evaluation should include long-horizon attacks, persistent memory attacks, malicious tool outputs, poisoned webpages, compromised plug-ins, and multi-agent propagation. Short single-turn jailbreak benchmarks underestimate risk because many agent failures emerge only after multiple tool calls and state updates.

6.5 Monitoring, red teaming, and governance

Monitoring closes the loop. Useful telemetry includes prompt hashes, retrieved source IDs, model version, tool-call arguments, policy decisions, refusal reasons, detector scores, human approvals, and anomalies in token usage or tool-call frequency. Privacy-sensitive logging should use minimization, access control, retention limits, and secure redaction.

Red teaming should be continuous rather than one-time. Regression suites should include known jailbreaks, prompt-injection patterns, RAG poisoning cases, privacy probes, tool misuse scenarios, and availability stress tests (Derczynski et al., 2024; Ganguli et al., 2022). Governance should define risk ownership, review processes for new tools, incident-response procedures, and user-facing disclosure. Security evaluations should be versioned because model and prompt updates can change behavior even when application code remains fixed.

7 Evaluation Methodology

Evaluation is a central weakness of current LLM-security research. Many papers report attack success rate on a small set of prompts, but deployment risk depends on attacker capability, tool privileges, task distribution, monitoring, and user behavior. A systematization should separate model-level metrics from system-level metrics.

7.1 Metrics

Common model-level metrics include attack success rate, refusal rate, harmful completion rate, toxicity score, privacy exposure, exact training-string recovery, membership-inference advantage, backdoor trigger success, clean-task accuracy, and transferability across models. System-level metrics include unsafe action rate, unauthorized data access, privilege-escalation distance, time to detection, rollback success, false positive rate of filters, user-task utility, cost amplification, and incident severity.

Attack success rate should be interpreted carefully. A jailbreak that produces unsafe text is not equivalent to an agent that sends an unauthorized email. Conversely, a low text-level harmfulness score may hide integrity failure if the model silently changes a database query. Security evaluations should therefore specify the protected asset, attacker goal, allowed actions, and success criteria.

7.2 Benchmarks and test design

Benchmarks should cover direct prompts, indirect content, RAG corpora, tool outputs, code repositories, multimodal inputs, persistent memory, and multi-turn sessions. They should include benign hard cases to measure over-refusal and utility loss. For RAG and agents, benchmark tasks should include realistic documents, permissions, and tool side effects. AgentDojo-like environments illustrate the importance of evaluating both task success and attack resistance (Zhang et al., 2024; Zhao et al., 2026).

Reproducibility is difficult because many leading models are closed, versioned silently, and protected by changing safety layers. Papers should report model name, date, version, decoding parameters, system prompts when publishable, prompt templates, tool schemas, detector thresholds, and source code. For closed models, repeated evaluation over time is necessary because behavior can drift.

7.3 Threat-model reporting

Every evaluation should state attacker knowledge and access. A black-box user who can only submit prompts is different from an attacker who can upload documents to a RAG corpus, control a webpage, edit a tool manifest, fine-tune an adapter, or tamper with model artifacts. The paper should also report whether the attack requires many queries, hidden text, model gradients, training access, or social engineering. Without this

reporting, comparisons across defenses are misleading. Table 5 lists our recommended reporting items for LLM vulnerability studies.

8 Risk Prioritization for Deployment

A key lesson from the lifecycle view is that vulnerabilities should not be prioritized only by their standalone attack success rate. A jailbreak that succeeds against an isolated chatbot may be less operationally dangerous than a lower-success-rate indirect prompt-injection attack against an enterprise agent with access to private documents, email, calendars, code repositories, or production systems. Deployment risk is therefore a function of the *model*, the *application stack*, the *available tools*, the *data context*, and the *authority delegated to the system*. This section provides a practical prioritization framework for translating the vulnerability taxonomy into deployment decisions.

8.1 A Deployment-Oriented Risk Model

For deployed LLM systems, risk should be evaluated at the level of a *model–application–authority tuple*, rather than at the level of the base model alone. We define a deployment scenario by

$$\mathcal{S} = (\mathcal{M}, \mathcal{C}, \mathcal{D}, \mathcal{T}, \mathcal{A}, \mathcal{H}),$$

where $\mathcal{M}$ is the model or model ensemble, $\mathcal{C}$ is the prompt and context-construction pipeline, $\mathcal{D}$ is the set of accessible data sources, $\mathcal{T}$ is the set of tools and external interfaces, $\mathcal{A}$ is the authority delegated to the system, and $\mathcal{H}$ represents human oversight and incident-response mechanisms. This formulation makes explicit why two deployments using the same model can have very different security profiles.

A practical prioritization score can be described as

$$R = f(E, F, I, D, O),$$

where E denotes exposure, F attacker feasibility, I impact, D defense maturity, and O observability. Exposure measures how many users, documents, tools, and external channels can reach the system. Feasibility captures the attacker's required knowledge, access, cost, and repeatability. Impact measures confidentiality, integrity, availability, safety, privacy, legal, and reputational consequences. Defense maturity captures whether controls are deterministic, tested, auditable, and resilient to adaptive attackers. Observability measures whether the organization can reconstruct what happened after a security event. This model is consistent with the broader adversarial-machine-learning view that risk depends not only on attack method, but also on attacker capability, system objective, and operational context (MITRE, 2024; OWASP GenAI Security Project, 2025; Vassilev et al., 2025).

8.2 Risk Drivers Across the LLM Application Stack

In operational settings, five risk drivers are especially important.

External authority. The most important question is what the system can do outside the chat window. A system that only generates text has limited direct side effects. A system that can send email, modify code, query databases, invoke cloud APIs, make purchases, operate robots, or run terminal commands has substantially higher agency risk. For this reason, tool access and delegated authority are often stronger predictors of risk than model size.

Untrusted context. Indirect prompt injection becomes possible when untrusted content is placed into the model context. This includes retrieved web pages, PDFs, emails, code comments, repository files, calendar descriptions, database records, browser content, tool outputs, and vector-database chunks (Greshake et al., 2023; Liu et al., 2023a; Xiong et al., 2024; Zhao et al., 2024). A deployment that mixes untrusted content with instructions and private data should be treated as high risk unless it uses strong provenance, isolation, and authorization controls.

Private and regulated data. Enterprise assistants, healthcare systems, financial systems, educational platforms, and legal tools often expose models to sensitive data. Risks include training-data memorization, retrieval leakage, embedding leakage, membership inference, prompt logging, cross-tenant data exposure, and unauthorized summarization (Carlini et al., 2021; Shokri et al., 2017). The priority of privacy controls increases when data is personally identifiable, legally protected, proprietary, or security-sensitive.

Irreversible or high-impact actions. Some actions are difficult to undo, such as sending confidential information, deleting files, merging code, changing access policies, executing financial transactions, publishing content, or triggering physical movement. These actions require stronger approval, simulation, transaction preview, logging, and rollback mechanisms.

Evidence after an incident. A system that cannot explain which model version, prompt, retrieved document, tool call, policy decision, or human approval produced an action is difficult to audit. Observability is therefore a security control, not only an engineering convenience. Logs should preserve security-relevant events while minimizing unnecessary retention of sensitive content.

8.3 Scenario-Oriented Risk Prioritization

Table 6 summarizes common deployment scenarios. The priority level is not absolute: it should be adjusted according to the sensitivity of the domain, the maturity of controls, and the degree of external authority. However, the table illustrates a recurring pattern: LLM risks become most urgent when untrusted content, private data, and external actions are combined.

8.4 Control Selection by Risk Driver

Risk prioritization should lead to concrete control selection. The most effective controls are often not better prompts, but stronger system boundaries. Prompt-level instructions are useful but should not be treated as the primary security mechanism when the system has access to private data or external tools.

Table 5: Recommended reporting checklist for LLM vulnerability studies.

Item	What to report	Why it matters
System boundary	base model, prompt layer, RAG, tools, memory, deployment stack	identifies what is actually being evaluated
Attacker capability	prompt access, corpus write access, tool-output control, fine-tuning access, artifact control	determines feasibility and relevance
Protected asset	private data, model weights, safety policy, tool action, availability, user trust	avoids vague claims of "security"
Success criterion	harmful text, data leakage, unauthorized action, cost increase, policy violation	makes attack success comparable
Utility metric	task accuracy, helpfulness, latency, cost, over-refusal, user satisfaction	detects defenses that simply break the system
Model/version details	model family, date, parameters when known, decoding, safety layer	supports reproduction and drift analysis
Context construction	system prompt, retrieved documents, memory policy, delimiters, tool schemas	critical for prompt-injection and RAG studies
Defense placement	pre-input, model, retrieval, output, tool boundary, monitoring, human approval	distinguishes probabilistic from deterministic controls
Reproducibility	code, prompts, datasets, random seeds, refusal classification rules	enables independent verification
Ethical controls	safe examples, disclosure process, harm minimization	reduces dual-use risk

8.5 Domain-Specific Considerations

Healthcare. Healthcare deployments combine high privacy requirements with high consequences for hallucinated or incomplete advice. The most important risks are leakage of protected health information, unsupported medical recommendations, biased triage, unsafe summarization of clinical records, and poor auditability. RAG sources should be curated and versioned; outputs should include provenance; and high-impact recommendations should remain under professional oversight. The model should assist documentation, retrieval, and patient communication rather than independently making diagnosis or treatment decisions.

Finance and insurance. Financial systems face risks involving private records, regulatory compliance, fraud, market manipulation, and unauthorized transactions. LLMs used for customer support, claims processing, investment research, or internal analytics should separate advice generation from transaction authority. Controls should include source provenance, suitability checks, human approval for financial actions, data-retention limits, and strong audit trails.

Education. Educational deployments involve student privacy, academic integrity, biased feedback, misinformation, and inappropriate delegation of grading authority. Systems should disclose limitations, avoid storing unnecessary student data, distinguish tutoring from assessment, and preserve instructor oversight. For grading or feedback systems, rubrics, provenance, and appeal mechanisms are important accountability controls.

Cybersecurity. LLMs can support defensive analysis, log triage, malware explanation, vulnerability management, and incident response, but they can also lower the barrier to abuse (Deng et al., 2023). The main deployment question is not whether the model can discuss security, but what tools, logs, credentials, and exploit-generation workflows it can access. Strong role-based access control, policy gating, controlled tool environments, and complete audit trails are essential.

Software engineering. Coding agents face a distinctive form of indirect prompt injection because repository content is both task data and a potential instruction carrier. Malicious instructions can appear in comments, documentation, tests, issue text, dependency metadata, build logs, or generated tool output. Secure design should treat repository content as untrusted unless explicitly trusted by the developer. Risk controls include sandboxed execution, dependency review, restricted file writes, approval before commits, and separation between code suggestion and code execution.

Enterprise knowledge management. Internal assistants often combine private documents, access control, RAG, and persistent memory. The main risk is that retrieval may bypass document permissions or mix information across users, teams, or tenants. Access control should be enforced before retrieval, not only after generation. Retrieved chunks should carry provenance and sensitivity metadata through the full pipeline.

Robotics and physical systems. The safety impact increases sharply when language plans control physical actions. A wrong instruction that would be merely inconvenient in a chatbot can become dangerous when translated into motion, navigation, manipulation, or equipment control. Systems require simulation, constrained controllers, runtime monitors, emergency stops, and human approval for risky operations. Natural-language plans should not directly map to actuator commands without safety validation.

Table 6: Deployment-oriented risk prioritization. Priority depends on the specific application, but this matrix summarizes common patterns.

Scenario	Typical priority	Dominant objective	Why	First controls to implement
Public chatbot without tools	medium	safety, abuse	large exposure, but limited external side effects	abuse monitoring, refusal calibration, rate limits, safety filters
Enterprise RAG assistant over private documents	high	confidentiality, integrity	private data and indirect prompt injection through documents	access control before retrieval, provenance labels, context isolation, logging
Coding assistant with repository and terminal access	high	integrity, agency	prompt injection in files can lead to code changes or command execution	sandboxing, least privilege, command approval, repository trust policy
Email/calendar/browser agent	high	privacy, agency	untrusted messages or webpages can trigger actions under user authority	tool scoping, human confirmation, untrusted-content handling, transaction logs
Fine-tuned domain model	medium–high	safety, privacy	private fine-tuning data and safety regression risk	data audit, privacy tests, safety regression, adapter/version control
Open model distribution	high	supply-chain integrity	many downstream users trust artifacts and loaders	signed weights, safe serialization, reproducible release, dependency scanning
Security-operation or penetration-testing assistant	high	safety, accountability	dual-use tasks and privileged context	role-based access, audit logs, policy gating, controlled tool environments

8.6 Practical Risk-Review Questions

Before deployment, a risk review should answer the following questions:

1. **Authority:** What external authority does the model or agent have, and which actions are irreversible or high impact?
2. **Trust boundaries:** What untrusted content can enter the prompt, retrieval context, memory, tool outputs, or system instructions?
3. **Data exposure:** What private, regulated, proprietary, or security-sensitive data can be retrieved, generated, stored, or logged?
4. **Tool containment:** Are tools allowlisted, scoped, sandboxed, and mediated by structured schemas and policy checks?
5. **Human oversight:** Which actions require human confirmation, and does the user see a faithful preview of the proposed action?
6. **Evaluation:** Has the system been tested against direct prompt injection, indirect prompt injection, jailbreaks, retrieval poisoning, privacy probes, and tool-misuse scenarios?
7. **Observability:** Can the organization reconstruct model version, prompt context, retrieved sources, tool calls, policy decisions, and approvals after an incident?
8. **Recovery:** Is there a rollback plan for model updates, poisoned documents, unsafe memories, leaked credentials, or harmful tool actions?

These questions often reveal that the highest-value controls are architectural rather than linguistic. Narrower privileges, clearer provenance, safer retrieval, deterministic tool boundaries, and better incident evidence usually reduce deployment risk more reliably than adding additional natural-language instructions to the prompt.

9 Open Problems

The preceding sections show that LLM vulnerabilities are not isolated defects of a model, but emergent properties of a lifecycle and application stack. A deployed system combines training data, alignment procedures, prompts, retrieval pipelines, memory stores, tools, human users, organizational policies, and software dependencies. This coupling creates several research challenges that cannot be solved by improving model refusal behavior alone. We organize the open problems into three groups: *foundational security problems*, which concern the boundary between language and computation; *system-design problems*, which arise when LLMs are connected to data and tools; and *assurance and governance problems*, which determine whether deployed systems can be evaluated, monitored, and repaired.

9.1 Compositional Security Guarantees

Current defenses rarely compose. A prompt-injection detector, a RAG filter, a safety classifier, and a tool schema may each perform well in isolation, yet fail when an attacker chains them across a long interaction. For example, a malicious document may first influence retrieval, then bias summarization, then

trigger a tool call, and finally hide evidence in a generated report. This type of cross-layer failure is difficult to capture with single-turn attack success rates.

The central open problem is how to provide security guarantees for an application whose planner is nondeterministic, whose inputs include adversarial natural language, and whose intermediate states are partially hidden. Formal methods, information-flow control, capability systems, typed tool interfaces, and runtime monitors are promising directions, but they remain difficult to integrate with probabilistic language models. Future work should move from evaluating isolated defenses to evaluating *defense compositions*. Useful questions include: whether a system preserves confidentiality under arbitrary retrieved text, whether an agent can invoke only authorized tools, and whether safety policies remain invariant under summarization, translation, compression, or multi-step planning.

9.2 Instruction–Data Separation

Indirect prompt injection exploits the absence of a reliable boundary between instructions and data. In conventional secure systems, code and data are separated by design; in many LLM applications, this boundary is blurred because both user commands and untrusted documents are represented as natural language in the same context window. Delimiters, role labels, and natural-language warnings are useful engineering practices, but they do not create a hard security boundary.

A major research direction is the design of stronger instruction–data separation mechanisms. Possible approaches include typed context regions, non-executable data blocks, provenance-aware attention mechanisms, structured intermediate representations, and external policy engines that decide which content may authorize an action. Another direction is to train or adapt models to respect context labels such as `instruction`, `untrusted document`, `tool output`, `memory`, and `policy`. The challenge is to make these labels semantically effective rather than merely decorative. A mature solution would allow an LLM to read untrusted content for information while preventing that content from changing goals, permissions, or tool policies.

9.3 Provenance-Aware RAG and Long-Term Memory

RAG and memory systems introduce persistent, externalized context. They improve factual grounding and personalization, but they also create new attack surfaces: poisoned documents, stale knowledge, cross-user leakage, embedding inversion, unauthorized retrieval, and memory manipulation. Current vector databases often optimize semantic similarity while carrying limited trust metadata. As a result, retrieved content may be relevant but unsafe, outdated, unauthorized, or adversarial.

Future RAG systems should preserve provenance across ingestion, chunking, embedding, indexing, retrieval, reranking, summarization, and answer generation. Open problems include embedding-level access control, poisoning-resistant retrieval, source conflict handling, freshness-aware generation, and verifiable deletion of memories. Memory systems require particular care because they can silently influence future sessions. Users and auditors should be able to inspect what is stored, why it was stored, when it expires, which future tasks may use it, and how it can be deleted. The long-term goal is not simply more accurate retrieval, but accountable retrieval with traceable authority.

9.4 Secure Agency and Tool Delegation

LLM agents require a theory of delegated authority. A user may ask an agent to plan a trip, summarize email, search a repository, or refactor code. The agent should infer helpful steps, but it should not invent new authority. This distinction is difficult because many useful tasks require planning, tool use, and interpretation of ambiguous intent.

Secure agency requires dynamic scoping of permissions to the user's current goal. Tool access should be narrow, temporary, auditable, and revocable. High-impact actions should require confirmation, and the confirmation should describe the actual external effect rather than merely restating the model's plan. For example, 'send this email to these recipients with this attachment" is a stronger confirmation than 'continue?" Evaluation should measure not only whether agents complete tasks, but also whether they avoid unsafe actions under malicious observations, poisoned tool outputs, or conflicting instructions. Future benchmarks should include realistic tool permissions, partial failures, deceptive documents, and multi-step attacks.

9.5 Privacy-Preserving Adaptation

Enterprises increasingly adapt LLMs using private documents, tickets, code, emails, logs, and user interactions. This creates privacy risks across both parametric and non-parametric components of the system. Fine-tuning may memorize sensitive examples; embeddings may leak semantic information; retrieval may cross tenant boundaries; logs may store private prompts; and model outputs may reveal information about training or retrieved data.

Differential privacy, federated learning, secure enclaves, synthetic data, redaction, and retrieval isolation provide partial solutions (Abadi et al., 2016; Dwork et al., 2006; McMahan et al., 2017). However, the open challenge is to preserve utility while providing measurable guarantees for the whole application stack. A privacy guarantee for model parameters alone is insufficient if the retrieval store, prompt logs, cache, or memory layer remains exposed. Future work should develop end-to-end privacy accounting for LLM applications, including model adaptation, embedding generation, retrieval, logging, and human feedback collection.

9.6 Robustness Under Distribution Shift and System Evolution

LLM applications evolve continuously. Models are updated, prompts are revised, retrieval corpora change, tools are added,

user behavior shifts, and attackers adapt. A defense that works at deployment time may degrade after a model update or after new documents enter the retrieval index. This creates a form of security distribution shift.

Open problems include safety regression testing, update-aware red teaming, continuous evaluation, and automatic detection of behavior drift. Model updates should be treated as security-relevant events, especially when the application has access to tools or private data. Similarly, retrieval-index updates should be tested for poisoning and permission errors. A mature deployment pipeline should evaluate not only model quality, but also whether existing safety, privacy, and tool-use guarantees still hold after each change.

9.7 Evaluation Realism and Reproducibility

Many current benchmarks underrepresent adaptive, long-horizon, stateful attacks. Real attackers can observe failures, revise prompts, poison future retrieval, exploit tool outputs, wait for maintenance windows, and chain small weaknesses across components. A single-turn jailbreak benchmark is therefore insufficient for evaluating deployed LLM systems.

Future benchmarks should include multi-session attacks, realistic permissions, benign hard cases, cost and latency constraints, multi-agent communication, retrieval poisoning, tool misuse, and adaptive attackers who know the defense. Evaluation should report not only attack success rate, but also false-positive rate, utility loss, cost, latency, user friction, and failure recoverability. Reproducibility is another open challenge because many studies depend on closed models, changing safety policies, and non-deterministic decoding. Shared benchmark harnesses, versioned prompts, fixed evaluation protocols, and transparent reporting checklists are needed for cumulative progress.

9.8 Multimodal and Cross-Channel Attacks

As LLM systems become multimodal, the attack surface expands beyond text. Instructions can be embedded in images, screenshots, audio, video frames, documents, tables, code, diagrams, or user-interface elements. A model may extract text from an image, summarize a PDF, interpret a webpage, or act on a screenshot. Each of these channels can carry untrusted instructions.

The open problem is to define security policies that survive cross-modal translation. A malicious instruction hidden in an image should not gain authority merely because it was converted into text by an OCR module or vision-language model. Similarly, a table cell, figure caption, metadata field, or audio transcript should carry provenance and trust labels after conversion. Future research should study multimodal prompt injection, cross-modal provenance, safe document parsing, and security-aware multimodal representation learning.

9.9 Human Factors, Usability, and Over-Reliance

LLM security is also a human factors problem. Users may over-trust fluent outputs, ignore warnings, approve unsafe tool calls, paste sensitive information, or misunderstand the system's authority. Conversely, overly strict defenses may create alert fatigue, reduce usefulness, or encourage users to bypass safeguards.

Open problems include designing effective confirmations, calibrated uncertainty displays, user-facing provenance, and warnings that are specific enough to change behavior. A useful confirmation should reveal the concrete consequence of an action, the data sources used, and the uncertainty or risk involved. More broadly, LLM systems should be designed so that users can understand what the system knows, what it can do, what it is not allowed to do, and when human expertise is required.

9.10 Incident Response, Auditing, and Governance

LLM incidents may involve prompts, retrieved documents, memories, model versions, tool calls, credentials, users, or third-party services. Organizations need runbooks for identifying affected sessions, revoking leaked credentials, quarantining poisoned documents, rolling back model versions, deleting unsafe memories, notifying users, and updating evaluation suites. Unlike traditional software incidents, the root cause may be a combination of natural-language input, model behavior, retrieval state, and tool execution.

Governance should specify who approves new tools, who owns safety regressions, who reviews model updates, how prompts and policies are versioned, and how incidents are documented. Auditing mechanisms should preserve enough evidence to reconstruct security-relevant events without retaining unnecessary sensitive content. Future work should develop standard incident taxonomies, evidence schemas, and post-incident evaluation methods for LLM applications.

10 Discussion

LLM vulnerabilities are often described as mysterious properties of neural networks, but many practical failures arise from familiar systems problems: weak input validation, confused-deputy behavior, excessive privilege, insecure supply chains, insufficient logging, and missing incident response. What is new is the interface: natural language can simultaneously express user intent, untrusted evidence, executable instructions, and social manipulation. This makes LLM systems especially vulnerable when developers rely on model judgment to enforce boundaries that should be enforced by architecture.

A lifecycle and application-stack view also helps avoid two common extremes. The first extreme is model fatalism: assuming LLMs are inherently unsecurable because prompt injection cannot be perfectly solved at the model level. The second is model optimism: assuming that better alignment or a stronger

system prompt will solve application security. The realistic position is layered. Models should become more robust, but applications must still constrain authority, preserve provenance, verify outputs, and monitor behavior.

The most important design shift is to treat LLMs as components inside security-critical systems, not as the security boundary themselves. The model can summarize, reason, draft, classify, and propose. It should not be the only component deciding whether untrusted content can override policy, whether a tool may execute, whether private data may leave a boundary, or whether a memory should persist.

11 Comparison with Existing Surveys and Frameworks

Table 7 clarifies novelty. We do not claim that each vulnerability is new. Instead, our contribution is a systems organization that maps attacks and defenses to the lifecycle and application stack. This framing is useful for practitioners because mitigation responsibility often belongs to different teams: data engineers, model trainers, safety teams, MLOps engineers, application developers, security teams, compliance officers, and human operators.

12 Conclusion

This survey reframed LLM vulnerabilities through a lifecycle and application-stack taxonomy. We organized attacks across data collection, pretraining, post-training alignment, packaging and supply chain, retrieval and memory, prompting and inference, tool/agent execution, and deployment/maintenance. This view shows that many risks are not isolated model failures but cross-boundary failures involving data provenance, instruction hierarchy, delegated authority, persistent state, and operational controls. We synthesized defenses into a defense-in-depth architecture that combines deterministic controls, model-level robustness, retrieval and memory governance, tool containment, monitoring, red teaming, privacy engineering, and incident response. The central lesson is that trustworthy LLM systems require both safer models and safer systems. Future work should move from isolated jailbreak demonstrations toward compositional security, provenance-preserving RAG, secure agency, realistic stateful benchmarks, and operationally grounded governance.

References

M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the ACM Conference on Fairness, Accountability, and Transparency*, pages 610–623, 2021.

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.

N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *USENIX Security Symposium*, pages 267–284, 2019.

N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, Ú. Erlingsson, A. Oprea, and C. Raffel. Extracting training data from large language models. In *USENIX Security Symposium*, pages 2633–2650, 2021.

X. Chen, C. Liu, B. Li, K. Lu, and D. Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

K. Chu. A systematic survey of security threats and defenses in LLM-based AI agents: A layered attack surface framework. *arXiv preprint arXiv:2604.23338*, 2026.

A. Dehghantanha et al. SoK: The attack surface of agentic AI – tools, and autonomy. *arXiv preprint arXiv:2603.22928*, 2026.

G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass. PentestGPT: An LLM-empowered automatic penetration testing tool. *arXiv preprint arXiv:2308.06782*, 2023.

L. Derczynski, E. Galinkin, J. Martin, S. Majumdar, N. Inie, et al. garak: A framework for security probing large language models. *arXiv preprint arXiv:2406.11036*, 2024.

E. Derner and K. Batistic. A security risk taxonomy for large language models. *arXiv preprint arXiv:2309.06899*, 2024.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, pages 265–284, 2006.

Table 7: Positioning of this survey relative to representative surveys and practitioner frameworks. Checkmarks indicate primary coverage; partial coverage indicates that the topic is discussed but not used as an organizing axis.

Work / framework	Lifecycle	RAG	Agents	Privacy	Supply chain	Defenses	Evaluation	Main distinction
LLM security/privacy surveys (Xu et al., 2025; Yao et al., 2024)	partial	partial	partial	✓	partial	✓	partial	broad attack catalogues and privacy/security synthesis
LLM trustworthiness/alignment surveys (Liu et al., 2023b; Naveed et al., 2023)	partial	partial	partial	partial		✓	✓	emphasize model capabilities, alignment, and evaluation
Agent-security surveys (Chu, 2026; Ferrag et al., 2025; Ling et al., 2026)	✓	✓	✓	partial	partial	✓	✓	focus on agentic workflows and protocols
OWASP LLM Top 10 (OWASP GenAI Security Project, 2025)	partial	✓	✓	✓	✓	✓		practitioner-oriented risk taxonomy
NIST AML taxonomy (Vassilev et al., 2025)	✓	partial	partial	✓	partial	✓	partial	general adversarial ML terminology and lifecycle stages
This survey	✓	✓	✓	✓	✓	✓	✓	unifies lifecycle stage, application stack, security objective, attacker capability, and defense layer

M. A. Ferrag, N. Tihanyi, D. Hamouda, L. Maglaras, A. Lakas, and M. Debbah. From prompt injections to protocol exploits: Threats in LLM-powered AI agents workflows. *arXiv preprint arXiv:2506.23260*, 2025.

M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

D. Ganguli, L. Lovitt, J. Kernion, A. Askell, Y. Bai, S. Kadavath, B. Mann, E. Perez, N. Schiefer, K. Ndousse, et al. Red teaming language models to reduce harms: Methods, scaling behaviors, and lessons learned. *arXiv preprint arXiv:2209.07858*, 2022.

Gemini Team. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *Proceedings of the ACM Workshop on Artificial Intelligence and Security*, 2023.

T. Gu, B. Dolan-Gavitt, and S. Garg. BadNets: Identifying vulnerabilities in the machine learning model supply chain. In *Machine Learning and Computer Security Workshop*, 2017.

D. Kang, X. Li, I. Stoica, C. Guestrin, M. Zaharia, and T. Hashimoto. Exploiting programmatic behavior of LLMs: Dual-use through standard security attacks. *arXiv preprint arXiv:2302.05733*, 2023.

V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. In *Conference on Empirical Methods in Natural Language Processing*, pages 6769–6781, 2020.

K. Kurita, P. Michel, and G. Neubig. Weight poisoning attacks on pre-trained models. In *Annual Meeting of the Association for Computational Linguistics*, pages 2793–2806, 2020.

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

Y. Ling, S. Yu, Z. Chen, and C. Fang. Toward secure LLM agents: Threat surfaces, attacks, defenses, and evaluation. *arXiv preprint arXiv:2606.10749*, 2026.

Y. Liu, G. Deng, Z. Xu, Y. Li, Y. Zheng, Y. Zhang, L. Zhao, T. Zhang, K. Wang, and Y. Liu. Prompt injection attack against LLM-integrated applications. *arXiv preprint arXiv:2306.05499*, 2023a.

Y. Liu, Y. Yao, J. Ton, X. Zhang, R. Cheng, Y. Klochkov, M. F. Taufiq, and H. Li. Trustworthy LLMs: A survey and guideline for evaluating large language models' alignment. *arXiv preprint arXiv:2308.05374*, 2023b.

B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. *International Conference on Artificial Intelligence and Statistics*, pages 1273–1282, 2017.

MITRE. MITRE ATLAS: Adversarial threat landscape for artificial-intelligence systems, 2024. URL https://atlas.mitre.org/. Accessed 2026-06-29.

M. Nasr, R. Shokri, and A. Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. *IEEE Symposium on Security and Privacy*, 2019.

H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*, 2023.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022.

OWASP GenAI Security Project. OWASP top 10 for large language model applications 2025, 2025. URL https://genai.owasp.org/llm-top-10/. Accessed 2026-06-29.

F. Perez and I. Ribeiro. Ignore previous prompt: Attack techniques for language models. *arXiv preprint arXiv:2211.09527*, 2022.

X. Qi, Y. Zeng, T. Xie, P.-Y. Chen, R. Jia, P. Mittal, and P. Henderson. Fine-tuning aligned language models compromises safety, even when users do not intend to! *arXiv preprint arXiv:2310.03693*, 2023.

Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023.

R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 2023.

R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy*, pages 3–18, 2017.

N. Stiennon, L. Ouyang, J. Wu, D. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33:3008–3021, 2020.

F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In *USENIX Security Symposium*, pages 601–618, 2016.

A. Vassilev, A. Oprea, A. Fordyce, and H. Anderson. Adversarial machine learning: A taxonomy and terminology of attacks and mitigations. NIST Trustworthy and Responsible AI Report NIST AI 100-2e2025, National Institute of Standards and Technology, 2025. URL https://csrc.nist.gov/pubs/ai/100/2/e2025/final.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh. Universal adversarial triggers for attacking and analyzing NLP. In *Conference on Empirical Methods in Natural Language Processing*, pages 2153–2162, 2019.

A. Wan, E. Wallace, S. Shen, and D. Klein. Poisoning language models during instruction tuning. *International Conference on Machine Learning*, 2023.

A. Wei, N. Haghtalab, and J. Steinhardt. Jailbroken: How does LLM safety training fail? *arXiv preprint arXiv:2307.02483*, 2023.

L. Weidinger, J. Mellor, M. Rauh, C. Griffin, J. Uesato, P.-S. Huang, M. Cheng, A. Glaese, B. Balle, A. Kasirzadeh, et al. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359*, 2021.

K. Xiong, X. Liu, P. Zhang, et al. Towards understanding the security risks of retrieval-augmented generation. *arXiv preprint arXiv:2404.13093*, 2024.

W. Xu et al. A survey of attacks on large language models. *arXiv preprint arXiv:2505.12567*, 2025.

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. ReAct: Synergizing reasoning and acting in language models. *International Conference on Learning Representations*, 2023.

Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang. A survey on large language model security and privacy: The good, the bad, and the ugly. *arXiv preprint arXiv:2312.02003*, 2024.

R. Zhang et al. AgentDojo: A dynamic environment to evaluate attacks and defenses for LLM agents. *arXiv preprint arXiv:2406.13352*, 2024.

W. Zhao, Z. Li, P. Zhang, and J. Sun. ClawGuard: A runtime security framework for tool-augmented LLM agents against indirect prompt injection. *arXiv preprint arXiv:2604.11790*, 2026.

Y. Zhao, C. Wu, B. Li, et al. Poisoning retrieval corpora for retrieval-augmented generation. *arXiv preprint arXiv:2402.07867*, 2024.

D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*, 2023.