Security in the Wild: An Empirical Analysis of LLM-Powered Applications and Local Inference Frameworks

1st Julia Gomez-Rangel
Department of Computer Science
Texas A&M University-Corpus Christi
Corpus Christi, USA
jgomezrangel@islander.tamucc.edu

2nd Young Lee
Department of Computational,
Engineering and Mathematical Sciences
Texas A&M University-San Antonio
San Antonio, USA
ylee@tamusa.edu

3rd Bozhen Liu
Department of Computer Science
Texas A&M University-Corpus Christi
Corpus Christi, USA
bozhen.liu@tamucc.edu

***Abstract*—The rapid rise of open-source applications and frameworks powered by large language models (LLMs) has introduced new and complex security risks. While recent studies have explored prompt injection, model misuse, and runtime vulnerabilities in isolated cases, the system-wide security risks of this ecosystem remain underexamined. In this paper, we present an empirical study of security advisories reported through GitHub for popular LLM-Powered Applications (LPAs) and their underlying Local Inference Frameworks (LIFs, such as llama.cpp and vLLM), aiming to surface system-wide security risks across the LLM software stack. We curate and analyze a dataset of 50 real-world vulnerabilities, classifying them by type, severity, and root cause. Our analysis reveals different risk profiles: LPAs tend to suffer from input-driven web vulnerabilities, while LIFs exhibit memory safety and dependency-related issues. We also identify common and unique characteristics of security vulnerabilities in LPAs and LIFs when compared to traditional open-source projects. Our findings highlight the urgent need for systematic security practices, better disclosure mechanisms, and lifecycle-aware defenses across the rapidly evolving LLM software stack.**

***Index Terms*—LLM-Powered Application, Local Inference Framework, GitHub Security Advisory, Open-Source Software Development**

I. Introduction

Large language models (LLMs) have transformed the landscape of software development, powering a new generation of intelligent, language-aware applications [1]. These LLM-Powered Applications (LPAs) span a wide range of use cases, including autonomous agents [2], [3], [4], intelligent chat interfaces [5], [6], and AI coding assistants [7], [8]. Their widespread availability through open-source ecosystems and their ease of customization have contributed to the rapid evolution of LLM software stack widely adopted by both researchers and practitioners [9], [10].

LPAs often access LLM services through remote APIs or LLM Inference Frameworks (LIFs, *e.g.*, `llama.cpp`) that allow developers to run LLMs efficiently on local hardware. These frameworks are increasingly integrated into LPAs, offering enhanced control, privacy, and reduced latency as well as increasing the complexity of open-source LLM-related software [11], [12].

Despite the utility of LPAs and LIFs, their rapid proliferation has introduced new and under-explored security risks [13], [14], [15]. LLMs inherently blur the boundary between data and code, making them vulnerable to prompt injection [16], code execution flaws [17], and insecure dependency use [18]. Several early works have examined vulnerabilities in LLM ecosystems. For example, LLMSmith [19] uncovered over a dozen remote code execution flaws in popular LLM-integrated tools and assigned 13 CVEs. Other efforts have studied LLM prompt injection [20], [21], adversarial prompting [22], or lifecycle-level risks across datasets, models, and interfaces [23]. However, these works either focus on the LLM model layer (*e.g.*, model training [24], jailbreaks [25]) or on earlier versions of LLM-integrated tools (released in 2023) that have since evolved significantly. Given the accelerated release cycle and community-driven growth of LLM ecosystems [26], [1], prior studies can quickly become outdated. Moreover, there is limited empirical analysis of vulnerabilities as reported and tracked by the open-source community, *e.g.*, through platforms such as GitHub Security Advisories [27]. Meanwhile, no prior study has offered a side-by-side comparison of LPAs and LIFs, or contrasted them with traditional open-source projects, using real-world, developer-facing data such as GitHub advisories and patch metadata, leaving system-level security dynamics in the LLM ecosystem underexplored.

To address this gap, we present an empirical study of GitHub-reported security issues in LPAs and LIFs, using the most current and up-to-date data available. We construct a curated dataset of 50 publicly disclosed vulnerabilities from GitHub Security Advisories collected from the most popular LPAs (>1k stars, actively maintained since 2023) and LIFs (most widely adopted by popular LPAs), together with their CVE links, and fix releases from project inceptions to June 2025. We analyze these issues across multiple dimensions, including vulnerability types, time to fix, affected components, and disclosure timelines. We also examine the factors contributing to the persistence of long-lived and unpatched vulnerabilities observed in our study dataset, and how these challenges hinder timely remediation. Our analysis contributes:

1) A dataset of 50 real-world security vulnerabilities from GitHub security advisories across popular open-source LPAs and LIFs;
2) The first comparative analysis of LPAs and LIFs using public vulnerability disclosures, highlighting differences in vulnerability types, severity levels, and remediation patterns;
3) The common and unique security characteristics of LPAs and LIFs in comparison with traditional open-source projects;
4) Recommendations for proactive security practices in the LLM software ecosystem, including prioritization, testing, and system-wide coordination;
5) The dataset, results, and source code required to reproduce our findings are publicly available [28].

II. Study Scope and Related Work

A. LLM-Powered Applications

Previous research has used the term "LLM-integrated application" to describe software systems where LLMs are embedded as auxiliary components or enhancements [29], [30], [21], [31]. This is also known as "LLM-integrated system" and "LLM-based application", where LLMs typically serve specific sub-tasks within broader applications that remain operational even in the absence of the LLM [32], [33].

Differently, this study focuses on a more recent category: LLM-Powered Applications (LPAs), a group of software that has emerged alongside the increasing maturity, accessibility, and practical utility of LLMs. We define an LPA as an application where the LLM is the core component that directly drives its primary functionality. In such systems, user interactions, application logic, or service outputs fundamentally rely on the LLM. Without the LLM, the application becomes largely non-functional or entirely inoperative.

B. LLM Inference Frameworks

Our pervious study [34] discovers that 61% of popular open-source LPAs support running LLMs locally through LLM inference frameworks (LIFs), such as Ollama [35], llama.cpp [36] and LocalAI [37]. Integrating LIFs into LPAs has already became one major way to leverage LLMs, while the other is through API service. The security issues introduced by APIs [38], [39] can compromise LPAs when user's information is exchanged through APIs, which has been addressed by extensive research [40], [41], [42], [43], [44] and industry tools [45], [46], [47]. However, there is still a lack of understanding regarding the security risks introduced by LIFs.

Moreover, adoption of LIFs by LPAs requires handling substantial heterogeneous computational resources [48] (*e.g.*, CPUs and GPUs), ensuring compatibility [49] and managing complex dependencies for seamless integration [50]. Moreover, developers face unique issues such as model migration issues [51], missing robust evaluation methods [52], [51] and addressing ethical concerns in the generated content [53]. However, we still have limited understanding of these inference frameworks serving as the fundamental infrastructure that enables LPAs to operate in offline and privacy-preserving environments. Hence, including LIFs in our study is essential for capturing the full landscape of LPA development.

C. Related Empirical Security Studies

Early Focus on RCE and Supply-Chain Risk. The first systematic vulnerability excavation in this area was LLM-Smith [19], which exposed over 20 vulnerabilities (including 19 Remote Code Execution issues) across 11 LLM-integrated frameworks and tools, leading to 13 CVEs. More recently, a survey study [23] explored supply-chain vulnerabilities in 75 LLM-related projects, analyzing security risks across 13 stages of the LLM lifecycle, highlighting insecure model downloads, unsafe deserialization and dependency issues.

Tool-Centric Audits but Little Ecosystem-Wide Data. Security tool builders have rushed in with adapted static analyzers (*e.g.*, Bandit's Python rulesets [54], Semgrep's custom LLM rules [55]) and purpose-built scanners (*e.g.*, Protect AI's LLM Guard [56]). OSV-Scanner [57], developed by Google, provides automated vulnerability detection based on the Open Source Vulnerabilities database [58], and is increasingly used to flag known issues in dependency chains. However, published evaluations either target a single framework family (*e.g.*, LangChain agents [59]) or use bespoke test harnesses divorced from the public disclosure record.

What is Still Missing? Few prior works have focused on how security vulnerabilities from real-world, open-source LPA and LIF projects are reported, discussed, and patched over time at the ecosystem level. Our study analyzes the security advisories from both LPAs and the LIFs they depend on, and provides the first side-by-side comparison of LPAs and LIFs, alongside traditional open-source projects, using real-world, developer-facing data. This makes this study both timely and necessary, which helps developers design future LLM-related software in a safe and secure way.

III. Methodology

A. Dataset Collection

We conducted a multi-phase data collection process of both automated and manual steps to ensure relevance, coverage and quality.

Collection of Popular LPAs and LIFs We first identified candidate LPA repositories on GitHub using the search API [72] using LLM-related keywords as topics. We used the keywords "llm", well-known LLMs (*e.g.*, "chatgpt") and their companies (*e.g.*, "openai"), as well as LLM-powered software (*e.g.*, "chatgpt-app") [1]. To focus on influential applications, we included only repositories with over 1,000 GitHub stars. We adopted a manual process to filter out these repositories that were not aligned with the scope of LPAs as described

[1]The full list of keywords can be found in our replica repository.

TABLE I: Statistics of LPAs and LIFs in Our Study.

Project	#Stars/#Forks	First Release (#Commits)	#Vul(#Un-patched)
		LPAs	
open-webui [60]	104k/14k	2024-02-22 (12,055)	3 (0)
NextChat [61]	85.1k/61.1k	2023-03-21 (3,082)	1 (0)
lobe-chat [62]	63.8k/13.3k	2023-07-18 (5,716)	5 (0)
ragflow [63]	60.9k/6.1k	2024-04-15 (3,480)	2 (1)
anything-llm [64]	47k/4.8k	2024-07-26 (1,500)	2 (2)
firecrawl [65]	43.5k/4.1k	2024-09-05 (3,678)	1 (0)
khoj [66]	30.6k/1.7k	2022-08-15 (4,915)	4 (0)
ChuanhuChatGPT [67]	15.4k/2.3k	2023-06-14 (1,258)	1 (0)
DocsGPT [68]	16.8k/1.7k	2023-03-03 (3,962)	1 (0)
onyx [69]	13.2k/1.8k	2024-08-01 (4,190)	1 (0)
		LIFs	
llama.cpp [36]	83.5k/12.5k	2023-03-18 (5,994)	9 (0)
vLLM [70]	53.2k/8.9k	2023-06-20 (7,910)	18 (1)
sagemaker-python-sdk [71]	2.2k/1.2k	2017-12-04 (4,235)	2 (0)

in Section II-A. To be specific, each author independently reviewed project descriptions, README files, dependencies, and in some cases the codebase or functionality. The individual results were then consolidated through discussion to reach consensus on a final curated list of 89 open-source LPA repositories, ensuring our dataset accurately represents true LPAs rather than other LLM-related applications (*e.g.*, finetuning framework, vector database).

We observed that 54 of the 89 LPAs in our dataset support running local LLMs. We applied the same manual process described above to identify popular open-source LIFs that are widely adopted by LPAs and show consistent contributions from the open-source community, ranked by usage in descending order: `llama.cpp` [36], vLLM [70] and `sagemaker-python-sdk` [71].

Collection of GitHub Security Advisories For each selected repository from the previous step, we extracted security-related data from the GitHub Security tab, which includes disclosed vulnerabilities, associated CVE IDs (or other identifiers when available), advisory summaries, severity ratings, and linked pull requests or releases addressing the issues. We supplemented this data with additional metadata such as issue creation and fix timestamps and affected package versions when disclosed. This concludes a set of 50 security advisories in our dataset.

We applied the same keyword search on NVD [73], but the result was less useful: most of the vulnerabilities overlapped with those in our dataset, while the others are either from close-source commercial projects, using non-English documentation, or addressing vulnerabilities from non-LPAs (*e.g.*, development frameworks for AI). Hence, we excluded NVD results from our study.

B. Data Processing

Each advisory in the dataset was manually verified and annotated with metadata such as vulnerability type, severity rating, project name, affected and patched version release dates, and presence of CWE classifications. This enriched dataset provides a detailed view into the real-world open-source security landscape of actively maintained LLM tools and systems.

TABLE II: Comparison of Estimated Time to Fix.

Type	#Patched	#Un-Patched	Time to Fix (Days) Avg.	Median	Min	Max
LPA	17 (5*)	3	234.12	41.00	0	819
LIF	29 (14*)	1	504.04	398.50	0	2340

*: the number of vulnerabilities with open-ended constraints.

For each vulnerability, we estimated the time between introduction and patch using release version metadata. Based on advisory content and commit-level inspection, we also recorded and determined whether the root cause of the vulnerability lies in the LLM itself, originates from a third-party dependency, or arises from the application's internal logic. This allowed us to distinguish between first-party and third-party sources of risk in our analysis [74]. The resulting dataset forms the basis of our descriptive statistics, comparative analysis, and fix-time measurements presented in the following sections.

IV. Study Results

A. Our Study Dataset

Table I provides an overview of the open-source LPAs and LIFs included in our analysis. For each project, we report the numbers of stars and forks, the release date of the first public version with the number of commits, the total number of reported vulnerabilities from GitHub security advisory, and how many of those remain unpatched at the time of writing. Although we omit certain popularity metrics in the table, all selected projects are actively maintained and widely used, as indicated by their GitHub visibility (*i.e.*, the numbers of stars, forks and commits). This summary sets the stage for our comparison analysis of vulnerability types, fix timelines, and security patterns across the open-source LLM software ecosystem.

Surprisingly, the majority of advisories (29 out of 50) originated from LIF projects, surpassing the 21 observed in LPAs. This finding contrasts with assumptions from previous work [75] that most risks would cluster around user-facing orchestration logic, where user inputs are parsed, validated, and routed to LLMs or downstream services. This also indicates a broader attack surface and greater scrutiny of frameworks.

B. Comparing LPAs and LIFs

Time-to-Fix Existing works often use fix latency to quantify responsiveness to vulnerabilities and difficulty of maintaining a project [76], [77], [78]. Differently, we do not use CVE/GHSA publication dates for temporal analysis due to a systemic delay in disclosure: the published date is always posterior to the actual patch. Instead, we estimate the time-to-fix window using the interval between when a vulnerable version was released (*i.e.*, affected version's release date) and when the patched version became publicly available (*i.e.*, patched version's release date).

When an advisory described the affected versions using open-ended constraints (*e.g.*, <v1.1.0), and no narrower introduction range was documented, we conservatively assumed the vulnerability was present since the project's first public release. This aligns with prior work in open-source software

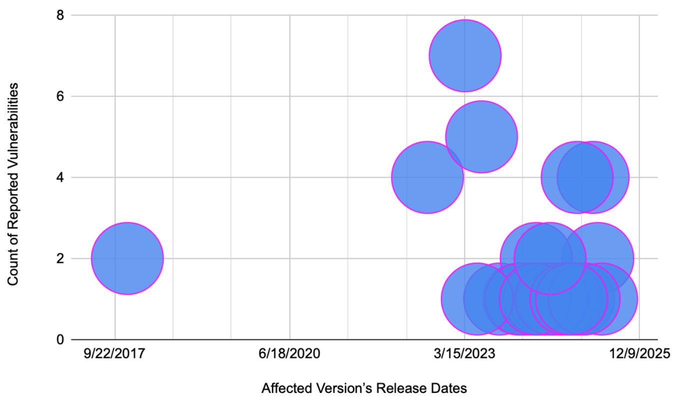

Fig. 1: The Distribution of Affected Version's Release Dates from Our Dataset (Each Blue Dot Represents A Vulnerability).

(OSS) vulnerability analysis where exact introduction commits are unavailable [79]. This method provides a meaningful approximation of the exposure window, *i.e.*, the time during which users may have unknowingly run vulnerable software, especially when disclosure metadata (*e.g.*, CVE publication dates) only reflect downstream reporting and not actual introduction.

Table II presents a comparative summary of estimated times to fix vulnerabilities in LPAs and LIFs, highlighting differences in remediation speed, range of delays, and the number of advisories analyzed per category. We noticed one outlier showed a negative fix window from an LPA (*i.e.*, -107 days from `onyx`) in our dataset and excluded it from this comparison. This outlier is probably due to inconsistencies in version tagging or backdated release metadata, highlighting the challenges in timely and formally documenting open-source software development and vulnerability report.

These results indicate that both LPAs and LIFs suffer from significant delays in patching introduced vulnerabilities, with LIFs showing longer average exposure windows due to the complexity of patching native code and tighter integration with system-level components. Although LPAs are often faster to patch (2X on average), they still exhibit long delays and metadata inconsistencies that complicate security lifecycle tracking. These findings suggest that vulnerability persistence is a systemic concern and mitigation will require structural improvements across both application- and framework-level development practices.

To understand whether security incidents are growing over time, we analyzed vulnerability introduction dates based on the release of affected versions. As shown in Figure 1, the data show a marked increase in issues from 2023 to present, with 22 vulnerabilities introduced in 2024 alone, and another 8 already introduced in 2025. This aligns with the timeline of open-source LLM proliferation and suggests that security debt accumulates quickly as new models and wrappers are adopted at scale. The trend reinforces the need for secure-by-default design and proactive security auditing in the LLM software lifecycle.

Common Vulnerability Types Figure 2 illustrates the distribution of vulnerability types observed in LPAs and LIFs, revealing distinct security characteristics between the two categories.

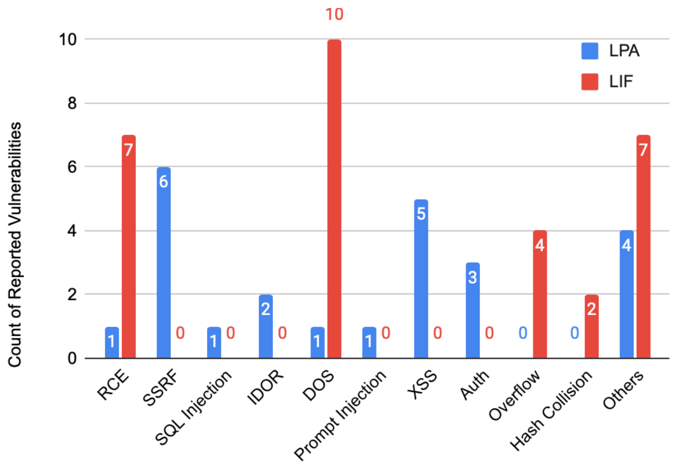

Fig. 2: Comparison of Vulnerability Types.

Among LPAs, the most frequent vulnerability type is Server-Side Request Forgery (SSRF), with 6 separate advisories. SSRF issues often emerge in retrieval-augmented generation (RAG) workflows, where unvalidated user input is passed into web request handlers for context augmentation. Other high-frequency patterns include Stored Cross-Site Scripting (XSS) with 5 advisories, insufficient authorization and authentication (Auth) with 3 cases and Insecure Direct Object References (IDOR) with 2 cases. These vulnerabilities reflect typical weaknesses in web-facing orchestration systems, especially those that rapidly integrate new LLM workflows without rigorous input sanitization or access control.

In contrast, LIFs face a very different class of vulnerabilities. The most reported issue type is Denial-of-Service (DoS), appearing 10 times, followed closely by Remote Code Execution (RCE) with 7 cases. These typically arise in the form of unsafe deserialization, malformed input handling, or resource exhaustion vulnerabilities in native code libraries. Additional patterns include hash collision attacks, overflow-based issues (*e.g.*, heap overflow, integer overflow), uninitialized memory use, and timing side-channel leaks (Others), all of which reflect the low-level nature and performance-critical focus of LIF implementations, often written in C or C++. In many cases, even minor errors in memory or buffer management can cascade into critical system-level vulnerabilities.

This bifurcation of risk profiles emphasizes the need for distinct security strategies: LPAs require hardening at the interface and orchestration levels (*i.e.*, input validation, endpoint restriction, sandboxing), while LIFs demand deeper investment in systems-level safe coding practices, fuzz testing, and memory-safe language adoption. It also suggests that generic LLM security scanners or rule engines may be insufficient unless tailored to these diverging threat models.

Common Root Causes Table III shows the root causes of reported vulnerabilities (identified by CWE IDs) from LPAs and LIFs, reinforcing the unique threat surfaces that each class

TABLE III: Frequency of CWEs in LPAs and LIFs.

CWE ID	CWE Count in LPA	LIF
Access Control		
CWE-284 Improper Access Control	1	0
CWE-639 Auth Bypass Through User-Controlled Key	2	0
Command Injection		
CWE-78 OS Command Injection	0	1
Data Integrity		
CWE-502 Deserialization of Untrusted Data	0	5
Exception Handling		
CWE-248 Uncaught Exception	1	0
Input Validation		
CWE-79 Cross-site Scripting (XSS)	1	0
CWE-80 Basic XSS	3	0
CWE-87 Improper Neutralization of Alternate XSS Syntax	1	0
CWE-89 SQL Injection	1	0
CWE-918 Server-Side Request Forgery (SSRF)	5	0
Memory Safety		
CWE-119 Improper Restriction of Operations within the Bounds of a Memory Buffer	0	2
CWE-122 Heap-based Buffer Overflow	0	1
CWE-123 Write-what-where Condition	0	1
CWE-125 Out-of-bounds Read	0	3
CWE-195 Signed to Unsigned Conversion Error	0	2
CWE-476 NULL Pointer Dereference	0	1
CWE-680 Integer Overflow to Buffer Overflow	0	1
Path Name/File Access		
CWE-22 Path Traversal	1	0
Sensitive Data Exposure		
CWE-200 Exposure of Sensitive Information to an Unauthorized Actor	2	0
URL Redirection		
CWE-601 Open Redirect	1	0
Unclassified or Missing CWE	3	15

of system exposes.

Vulnerabilities in LPAs are heavily concentrated around input validation and access control issues. To be specific, LPAs account for all reported cases of XSS (CWE-79, CWE-80), SQL injection (CWE-89), and SSRF (CWE-918), as well as access control failures (CWE-284 and CWE-639). These vulnerabilities originate from the application-layer complexity of LPAs, which often expose HTTP APIs, dynamically handle user input, and route data through prompt templates or external plugins, making them susceptible to injection and access flaws.

Differently, vulnerabilities in LIFs are dominated by memory safety and data deserialization issues. The most frequent root causes include CWE-502, CWE-119 and CWE-125, along with related overflow and pointer errors. These weaknesses are common in low-level systems implemented in C/C++, such as `llama.cpp` and `vLLM`, which prioritize performance and minimal dependencies but often lack memory safety guarantees.

In terms of severity distribution, LPAs accounted for eight high severity, seven moderate, four critical, and two low severity advisories. LIFs showed a larger cluster of moderate severity issues, with thirteen moderate, ten high, and four critical cases, along with two low severity reports. This distribution reinforces the need for more rigorous hardening in LIFs, especially at the systems level where low-level flaws may have far-reaching impact.

TABLE IV: Vulnerabilities Associated with Multiple CWE IDs.

Project	CVE ID	Associated CWE IDs
NextChat	CVE-2024-38514	CWE-79, CWE-918
llama.cpp	CVE-2025-53630	CWE-122, CWE-680
	CVE-2025-52566	CWE-119, CWE-195
	CVE-2025-49847	CWE-119, CWE-195

Unclassified or Missing CWE We further analyzed the 15 out of 29 LIF vulnerabilities without identified CWE IDs, and discovered several common themes stemming from emerging architectural patterns in LIFs, as shown below:

- *Abuse of LLM-specific logic* such as prefix caching, guided decoding, prompt chunk, introduce new classes of DoS, timing side channels, and data leakage tied specifically to LLM usage patterns, not traditional program logic.
- *Improper validation or parsing of complex inputs* such as malformed regex, invalid multimodal placeholders, and malicious JSON schemas, often manifest in crashes, DoS, or unpredictable model behavior, which are specific to LLM pipelines rather than web or service code.
- *Unsafe memory operations or deserialization* such as buffer overflows, null pointer dereferences, arbitrary address read and write, though can be mapped to CWEs, remains unclassified due to lack of fine-grained root cause analysis in reports or insufficient CWE documentation for LLM contexts.

Besides, many issues arise from distributed inference setups, GPU memory sharing, or structured output features. They don't map cleanly to traditional CWEs, which were designed for web and system software, not machine learning and LLM infrastructure. These unclassified issues reveal that LLM-related software introduces new security concerns that escape traditional classification schemes. Existing CWEs rarely account for LLM-specific control and data flows, inference-specific resource and state management, as well as distributed, multi-modal, or prompt-driven execution models. Hence, security tooling and taxonomies need to evolve to better capture emerging risks in LLM infrastructure.

Compound Vulnerabilities and Root Causes Table IV lists four vulnerabilities associated with more than one CWE ID. These cases often involve compound failure conditions that span multiple layers of a system. For example, CVE-2024-38514 in `NextChat` is linked to both XSS (CWE-79) and SSRF (CWE-918), suggesting that unsanitized user input can simultaneously affect client-side rendering and backend request logic. Such an overlap demonstrates how LPAs can exhibit multifaceted security exposures from a single point of failure due to its dynamically generated content and interaction

TABLE V: LLM-Specific Vulnerabilities in LPAs

Project	CVE/GHSL ID	LLM-Related Impact	Root Cause	Severity	Patched?
anything-llm	GHSL-2025-056	Ollama Token Leak	Missing Authentication Checks	High	Yes
lobe-chat	CVE-2024-24566	Unauthorized Access to Chat	Missing Authorization Checks	Moderate	Yes
ChuanhuChatGPT	CVE-2023-34094	API key leak	Unauthorized configuration file access	High	No

with external services.

The `llama.cpp` project contains three vulnerabilities with dual CWE classifications, including heap overflows (CWE-122), integer overflows (CWE-680), generic memory and type errors (CWE-119 and CWE-195). These issues reflect the complexity and risk of building high-performance inference engines in low-level languages such as C and C++. In such environments, minor arithmetic errors or boundary miscalculations can lead to memory corruption or exploitable behavior, which are exacerbated by the performance optimizations typically required for local LLM inference.

Vulnerabilities associated with multiple CWEs are particularly challenging to detect using static or rule-based tools, as they do not fit neatly into a single vulnerability category. Their presence reinforces the need for comprehensive security auditing practices that consider both surface-level and deep implementation flaws. For the LLM ecosystem, where application logic often includes complex routing, dynamic input composition, and memory-intensive operations, such compound vulnerabilities highlight the value of integrated security reviews that span both the application and infrastructure layers.

LLM-Specific Vulnerabilities in LPAs Among the above reported vulnerabilities, we classify certain issues as "LLM-specific" based on whether their security consequences directly affect the functionality of the LLM or compromise the confidentiality or integrity of LLM-related data or behavior. Based on this criterion, three vulnerabilities were identified as LLM-specific as shown in Table V. These cases underscore how traditional security flaws can have LLM-specific consequences when model components are deeply integrated into application workflows. The first instance involved token leakage from the Ollama backend, where weak configuration controls led to unintended exposure of credentials used in local inference. The other two vulnerabilities enabled unauthorized access to user chat sessions and API key leakage, caused by missing or insufficient access control enforcement across session boundaries and configuration files.

Importantly, in all three cases, the LLM-related security impact was a side effect of conventional weaknesses. This highlights a critical blind spot: when LLMs are embedded into applications, even small lapses in control logic or infrastructure setup can cascade into high-impact risks involving sensitive data leakage or LLM misuse.

These findings emphasize the urgent need for a systematic and layered security approach for LPAs. Guarding against LLM-specific threats cannot rely solely on traditional web security practices. Developers must adopt LLM-aware safeguards, including secure prompt routing [80], authenticated model access [81], bounded output control and token management policies [82] that explicitly account for model interaction surfaces. As LLMs become more tightly coupled with user interfaces and application logic, securing them demands a defense-in-depth strategy that holistically addresses both the conventional attack surface and the unique behaviors introduced by generative models.

Dependency-Related Vulnerabilities LIFs exhibited greater vulnerability to dependency-related risks, with 2 confirmed cases caused by third-party packages or transitive library flaws. No such cases were identified in LPAs. The difference reflects the fact that LIFs rely more heavily on platform-level and performance-optimized packages, often with bindings to native libraries. These dependencies may not be automatically tracked by vulnerability scanners such as GitHub Dependabot [83] or OSV-Scanner, leading to long windows of unpatched risk. Moreover, this finding is concerning, as LIFs are reused extensively in the LLM stack. A single vulnerable library in an LIF could cascade downstream across dozens of LPA projects without detection, making dependency hygiene in frameworks a critical blind spot.

Unpatched Vulnerabilities and Patch Challenges At the time of writing, 4 out of the 50 vulnerabilities in our dataset remained unpatched, leaving them persistently exposed in public releases. According to GitHub advisories and CVE records, all four were classified as either high or critical severity. These vulnerabilities are often tightly coupled with core architectural components of the affected systems, which makes remediation especially difficult.

The first unpatched case is an SQL injection (CVE-2025-27135), resolving the issue requires not just input sanitization but often a complete restructuring of how queries are built and executed [20]. If the application dynamically constructs queries from user input without strict abstraction layers (*e.g.*, through raw string interpolation), then fixing the vulnerability may involve deep changes to the application's data handling logic. Moreover, any refactoring must preserve existing functionality across varied use cases, which increases the burden and risk of introducing regressions.

The second unpatched vulnerability combines sensitive data exposure, insecure endpoint access and broken access control (GHSL-2025-056). Mitigating this issue requires not only securing individual API endpoints but also auditing authentication mechanisms, permission checks, and data filtering logic. Projects that were not initially designed with strict access controls may lack a centralized security enforcement point, indicating that the problem is distributed across multiple modules [84]. This fragmentation complicates both detection and remediation, which may require architectural redesign [85].

The third unpatched case is an unauthenticated DoS attack (CVE-2024-22422), which also presents a difficult fix, especially if it exploits resource exhaustion in unauthenticated request flows. Applying rate limits, request validation, or authentication gating can reduce the risk, but such changes must be implemented with care to avoid degrading performance or blocking legitimate users [86]. For LLM-powered systems, where requests can involve expensive operations (*e.g.*, embedding generation or model inference), even seemingly harmless inputs can lead to costly workloads [87]. This makes DoS mitigation a balance between resilience and usability.

The last unpatched case (CVE-2025-30165) has detailed documentation from the project maintainer explicitly stated that a patch was unnecessary [88]. The rationale provided was that the vulnerability could only be triggered in an uncommon deployment pattern or in a retired version that is no longer used by default. This highlights the subjectivity involved in patch triage decisions, especially in community-led or volunteer-maintained open source projects [89]. However, from a security consumer perspective, even low-severity vulnerabilities should ideally be mitigated, especially when the LPA may be deployed in unexpected or uncontrolled environments.

In summary, these unpatched issues exemplify the types of vulnerabilities that are not only high-impact but also high-friction to fix, often requiring structural changes, performance trade-offs, or complex validation workflows.

Reporting Quality The advisories for both LPAs and LIFs in our dataset were generally well documented, though differences in reporting style and completeness were evident. 86% LPA vulnerabilities included CWE mappings, while 52% LIF advisories lacked this classification. Most LPA and LIF advisories are consistent with structured CVE standards and often contained technical details such as exploit steps or buffer boundaries, while some omit clear mitigation steps or relying on vague severity descriptors such as “high” or “moderate” without further context.

V. Common and Unique Risks in LPAs and LIFs

This section analyzes well-established security flaws in LPAs and LIFs that mirror those long observed in traditional OSS, alongside emerging vulnerability patterns that are unique to the LLM ecosystem.

A. Vulnerability Types and Patch Delays

In traditional OSS projects hosted on GitHub, patch delays tend to be much shorter, averaging 40.46 days, with over 80% of CVE–branch pairs eventually patched [90]. Our study shows that LPAs and LIFs experience considerably longer delays, averaging 234 days for LPAs and 504 days for LIFs, with 8% of vulnerabilities still unpatched. Meanwhile, large OSS projects such as Chromium and OpenSSL demonstrate even longer timelines, requiring about 2 years and 7 years on average, respectively, to deploy fixes [79].

The vulnerability types observed in LPAs and LIFs largely overlap with those found in traditional OSS projects [91], including CWE-79 and CWE-22. This suggests that LPAs and LIFs inherit many of the foundational risks seen in traditional OSS. Meanwhile, LPAs are more prone to integration-level vulnerabilities, such as CWE-918. LIFs exhibit more low-level memory vulnerabilities, tied to performance-optimized native code (*e.g.*, C/C++). Hence, LLM-related software does not introduce entirely new vulnerability classes, however, they do amplify existing risks and demand tailored defenses at both application and infrastructure levels.

B. Emerging Patterns in LLM Ecosystem

From the 50 real-world vulnerabilities from LPAs and LIFs, we observed recurring patterns that reveal both software weaknesses amplified by the LLM context and risks that arise uniquely from it.

Emergence of Model-Centric Exploits One of the most distinctive patterns we observed from real vulnerabilities in LIFs is the rise of model-centric exploits. Unlike traditional OSS, where vulnerabilities often stem from API misuse or network exposure, LIFs expose new risks through the way they parse, load, and execute model files (*e.g.*, GGUF, safetensors, or custom weight formats). Attackers can craft malicious model files that exploit weaknesses in parsing routines, leading to heap overflows, out-of-bounds reads, or arbitrary code execution (observed in `llama.cpp` and vLLM). This turns what should be a trusted model artifact into a powerful attack vector. This is a *novel attack surface* because traditional OSS projects without LLM (or machine learning) components typically do not treat model weights or vocabulary files as untrusted input. The challenge is compounded by the fact that many LIFs are designed to load community-contributed or downloaded models, significantly expanding the attack surface. Our findings highlight that the simple act of running inference on a compromised model can yield outcomes ranging from DoS to RCE, underscoring how tightly bound LIFs are to model format integrity and the urgent need for hardened parsing logic.

General Weaknesses With LLM-Specific Consequences A second recurring pattern in our dataset is that traditional security flaws manifest in LLM-specific ways, amplifying their consequences. Many vulnerabilities in LPAs and LIFs share root causes with common OSS flaws, such as missing authentication and insufficient access controls. However, in LLM-powered context, these flaws cascade into LLM-specific risks. For example, a missing authorization check (observed in `lobe-chat` and `onyx`) that might otherwise expose a configuration file in a web app can, in an LPA, expose API keys that grant unrestricted access to an LLM backend, enabling attackers to exfiltrate sensitive user prompts or execute arbitrary model queries. Similarly, a DoS vulnerability triggered by malformed input can not only crash a worker but also interrupt ongoing inference sessions across multiple users, leading to more disruptive outcomes than in non-LLM systems. These cases show that while the underlying weakness was not new, the presence of an LLM at the center of the application dramatically raised the stakes, turning otherwise ordinary flaws into high-severity risks that directly compromise model-driven functionality. This reinforces the idea that LPAs inherit old problems but magnify them in ways that security teams may not anticipate.

VI. Implications

Our analysis of security advisories in LPAs and LIFs reveals a maturing but uneven security landscape. While a growing number of projects are adopting formal reporting practices through GitHub's security tab, the data also expose persistent gaps in patch timeliness, inconsistent documentation, and the absence of preventive mechanisms. These findings carry several important implications for the open-source LLM software community and suggest opportunities for improvements.

Proactive Vulnerability Detection and Prioritization One of the clearest takeaways from our study is the long time-to-fix window observed in both LPAs and LIFs, with average delays exceeding 200 days and some serious vulnerabilities remaining unpatched for years. Although reactive disclosure workflows are beginning to take hold, they remain insufficient in isolation. Given the security-critical nature of many LLM deployments ranging from enterprise chat systems to on-device assistants [7], [12], [3], [92], projects must adopt proactive vulnerability detection.

To support these efforts, there is a need for clearer triage processes and tagging standards. Vulnerabilities that span multiple layers (*e.g.*, orchestration logic, model behavior, and system-level execution) require consistent labeling and categorization to prioritize remediation. Our study shows that some vulnerabilities remained unpatched not because they were undetected, but because their risk was downplayed or a huge effort is required from maintenance team. Besides, standardizing how LLM-specific risks are reported and interpreted will enable better prioritization and more responsible downstream reuse.

CI for Security Enforcement We did not observe any security-focused CI pipelining (*e.g.*, GitHub Actions) from our studied LPAs and LIFs, suggesting that LLM-related projects lack tight integration between security mechanisms and CI workflows. Even among popular repositories, patches were often released manually and months after the initial issue was disclosed [93]. This delay is partially due to fragmented development processes and the absence of security enforcement gates in building pipelines [94].

To reduce exposure windows, projects should incorporate security checks directly into CI workflows. For example, input validation logic should be tested against prompt injection and command injection patterns as part of test suites. Dependency updates should trigger security scans, and results from CVE databases or GitHub advisories should be surfaced automatically during pull requests. Automating such checks can dramatically reduce remediation delays and prevent regressions.

Toward A Systematic Approach for the LLM Software Stack Our study calls attention to the need for a holistic and layered security strategy that spans the full LLM ecosystem. LPAs sit atop an increasingly complex stack that includes LIFs, system libraries, model files, and orchestration agents [95], [96]. Our study shows that vulnerabilities can arise at any layer through insecure API access, unsafe user/model behavior or flawed low-level memory handling. Hence, security efforts must move beyond individual applications and address the stack as a whole.

We advocate for the development of LLM-stack-aware security analysis tools that trace vulnerabilities across abstraction boundaries. For example, if an LPA uses an LIF vulnerable to command injection, that risk should be surfaced in the LPA's own advisory or dependency dashboard. Similarly, a centralized platform for cross-project vulnerability tracking could help detect when the same issue recurs across different agent frameworks or inference backends. Moreover, the growing reuse of patterns, such as RAG [97] or plug-in style agents [98], means that new vulnerabilities may propagate rapidly unless mitigated systematically. Establishing shared security guidelines, reusable defense patterns, and standardized audit frameworks will be key to protecting the broader LLM ecosystem.

VII. Threats to Validity

Construct Validity concerns whether the patterns we analyzed are accurately represented in our data, for example, classification of vulnerability types for LPAs and LIFs. To mitigate this, we manually examined documentation and metadata on our selected LPAs and LIFs with their GitHub security advisories, using predefined criteria for labeling and categorizing issues. While manual analysis enabled fine-grained inspection, it also introduces subjectivity, especially in ambiguous or undocumented cases. We mitigated this risk through consistent labeling guidelines and cross-validation across multiple data sources by multiple experts, but some interpretation bias may remain. In addition, we note the possibility that the four unpatched vulnerabilities may have been quietly fixed without update or disclosure [99], [100], which could affect the validity of our study.

Internal Validity refers to whether our conclusions about patterns and relations are sound and not confounded by uncontrolled factors. Our identification of vulnerabilities and patterns was based on the recurring presence of these issues across diverse repositories. However, since our data was filtered through manual inspection, errors in classification or under-representation of niche project types may affect the strength of our inferences. Moreover, in identifying common patterns, we based our findings on observable patterns in GitHub security advisory discussions and project behaviors. To reduce bias, we applied a consistent classification scheme and compared repeated issues across projects. Finally, not all of the observed patterns and relations can be definitively confirmed without direct evidence from maintainers.

External Validity addresses how well our findings generalize beyond our sampled dataset. The 89 LPAs were selected using GitHub search API, which may miss repositories using different naming conventions on topics. To mitigate this, we first collected representative keywords commonly used in LPA GitHub repositories and then used a custom scraper to improve the coverage and completeness. However, our dataset focuses on open-source LPAs and may not capture the features of commercial LPAs. Similarly, the selection of LIFs and GitHub security advisories we studied are primarily open-source and may not reflect the complexity or optimization of closed-source apps and frameworks. Even though our findings are highly relevant to open-source LLM development ecosystem, generalization to all LPA and LIF development should be made with caution.

VIII. Conclusions

Our study presents a comparison analysis of publicly disclosed security vulnerabilities across a representative set of open-source LPAs and LIFs. By mining and analyzing GitHub Security Advisories, we provide the first empirical snapshot of real-world LLM software security as reflected in developer-facing disclosures.

Our findings reveal distinct vulnerability profiles for LPAs and LIFs: LPAs tend to suffer from web-facing and input validation issues such as SSRF, XSS, and access control flaws, while LIFs are more prone to memory safety vulnerabilities and deserialization risks common in low-level systems code. We also identify unique security risks stemming from LPAs and LIFs, including new attack surfaces formed by LIFs and models, as well as LLM-specific consequences arising from general security weaknesses. In addition, we observe substantial variability in patching practices, with some high-severity vulnerabilities remaining unpatched for extended periods due to architectural entanglement or limited maintenance resources. Notably, several LPA vulnerabilities involved LLMs only indirectly, highlighting the complexity of securing entire software stacks that integrate LLMs.

The persistence and nature of these security issues underscore the need for proactive vulnerability detection, consistent triage practices, and better coordination across the LLM software ecosystem. Moving forward, we advocate for tighter integration of security scanning and policy enforcement into LLM-related CI/CD pipelines, along with a more systematic approach to securing not just LPA interfaces but also the underlying inference frameworks on which they depend. Our dataset and analysis aim to serve as a foundation for more targeted tools, benchmarks, and defense mechanisms tailored to the emerging LLM software landscape.

Acknowledgment

This work was supported by the CAHSI-Google Institutional Research Program; we express our sincere appreciation for their support.

References

[1] "Here are 18,369 public repositories matching this topic llm — github.com," https://github.com/topics/llm, 2025, [Accessed 25-04-2025].

[2] "Significant-Gravitas/AutoGPT: AutoGPT is the vision of accessible AI for everyone, to use and to build on. Our mission is to provide the tools, so that you can focus on what matters. — github.com," https://github.com/Significant-Gravitas/AutoGPT, [Accessed 17-07-2025].

[3] "reworkd/AgentGPT: Assemble, configure, and deploy autonomous AI Agents in your browser. — github.com," https://github.com/reworkd/AgentGPT, [Accessed 17-07-2025].

[4] "Microsoft Copilot: Your everyday AI companion — copilot.microsoft.com," https://copilot.microsoft.com, [Accessed 04-06-2024].

[5] "mckaywrigley/chatbot-ui: AI chat for any model. — github.com," https://github.com/mckaywrigley/chatbot-ui, [Accessed 17-07-2025].

[6] "lm-sys/FastChat: An open platform for training, serving, and evaluating large language models. Release repo for Vicuna and Chatbot Arena. — github.com," https://github.com/lm-sys/FastChat, [Accessed 17-07-2025].

[7] "GitHub Copilot: Your AI pair programmer — github.com," https://github.com/features/copilot, 2025, [Accessed 17-07-2025].

[8] "Cursor - the ai code editor." https://cursor.com/en, 2025, [Accessed 17-07-2025].

[9] Z. Liao, M. Antoniak, I. Cheong, E. Y.-Y. Cheng, A.-H. Lee, K. Lo, J. C. Chang, and A. X. Zhang, "Llms as research tools: A large scale survey of researchers' usage and perceptions," *arXiv preprint arXiv:2411.05025*, 2024.

[10] D. Russo, "Navigating the complexity of generative ai adoption in software engineering," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 5, pp. 1–50, 2024.

[11] X. Wang, Z. Tang, J. Guo, T. Meng, C. Wang, T. Wang, and W. Jia, "Empowering edge intelligence: A comprehensive survey on on-device ai models," *ACM Computing Surveys*, vol. 57, no. 9, pp. 1–39, 2025.

[12] B. A. Sokhansanj, "Local ai governance: Addressing model safety and policy challenges posed by decentralized ai," *AI*, vol. 6, no. 7, 2025. [Online]. Available: https://www.mdpi.com/2673-2688/6/7/159

[13] B. C. Das, M. H. Amini, and Y. Wu, "Security and privacy challenges of large language models: A survey," *ACM Computing Surveys*, vol. 57, no. 6, pp. 1–39, 2025.

[14] M. A. Ferrag, N. Tihanyi, D. Hamouda, L. Maglaras, and M. Debbah, "From prompt injections to protocol exploits: Threats in llm-powered ai agents workflows," *arXiv preprint arXiv:2506.23260*, 2025.

[15] Z. Lin, W. Ma, T. Lin, Y. Zheng, J. Ge, J. Wang, J. Klein, T. F. Bissyandé, Y. Liu, and L. Li, "Open source ai-based se tools: Opportunities and challenges of collaborative software learning," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–24, 2025.

[16] "Prompt injection attacks and defenses in llm-integrated applications," *arXiv preprint arXiv:2310.12815*, 2023.

[17] T. Coignion, C. Quinton, and R. Rouvoy, "A performance study of llm-generated code on leetcode," in *Proceedings of the 28th international conference on evaluation and assessment in software engineering*, 2024, pp. 79–89.

[18] S. Goetz and A. Schaad, ""you still have to study"–on the security of llm generated code," *arXiv preprint arXiv:2408.07106*, 2024.

[19] T. Liu, Z. Deng, G. Meng, Y. Li, and K. Chen, "Demystifying rce vulnerabilities in llm-integrated apps," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 1716–1730.

[20] "From prompt injections to sql injection attacks: How protected is your llm-integrated web application?" 2023.

[21] "Prompt injection attack against llm-integrated applications," 2024.

[22] A. Kumar, C. Agarwal, S. Srinivas, A. J. Li, S. Feizi, and H. Lakkaraju, "Certifying llm safety against adversarial prompting," *arXiv preprint arXiv:2309.02705*, 2023.

[23] S. Wang, Y. Zhao, Z. Liu, Q. Zou, and H. Wang, "Sok: Understanding vulnerabilities in the large language model supply chain," *arXiv preprint arXiv:2502.12497*, 2025.

[24] K. Wang, G. Zhang, Z. Zhou, J. Wu, M. Yu, S. Zhao, C. Yin, J. Fu, Y. Yan, H. Luo *et al.*, "A comprehensive survey in llm (-agent) full stack safety: Data, training and deployment," *arXiv preprint arXiv:2504.15585*, 2025.

[25] A. Wei, N. Haghtalab, and J. Steinhardt, "Jailbroken: How does llm safety training fail?" *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[26] X. Chen, C. Gao, C. Chen, G. Zhang, and Y. Liu, "An empirical study on challenges for llm application developers," *ACM Trans.*

Softw. Eng. Methodol., Jan. 2025, just Accepted. [Online]. Available: https://doi.org/10.1145/3715007

[27] "GitHub Advisory Database — github.com," https://github.com/advisories, [Accessed 18-07-2025].

[28] "apace-lab/Security_of_LPA_and_LIF-AIware2025 — github.com," https://github.com/apace-lab/Security_of_LPA_and_LIF-AIware2025, [Accessed 29-09-2025].

[29] I. Weber, "Large language models as software components: A taxonomy for llm-integrated applications," 2024. [Online]. Available: https://arxiv.org/abs/2406.10300

[30] J. Evertz, M. Chlosta, L. Schönherr, and T. Eisenhofer, "Whispers in the machine: Confidentiality in llm-integrated systems," 2024. [Online]. Available: https://arxiv.org/abs/2402.06922

[31] O. Topsakal and T. C. Akinci, "Creating large language model applications utilizing langchain: A primer on developing llm apps fast," in *International Conference on Applied Engineering and Natural Sciences*, vol. 1, no. 1, 2023, pp. 1050–1056.

[32] "Tone Detector and Tone Suggestions — Grammarly — grammarly.com," https://www.grammarly.com/tone\#, [Accessed 25-04-2025].

[33] Instacart, "Unlocking Efficiency: How Ava Became Our AI Productivity Partner — instacart.com," https://www.instacart.com/company/how-its-made/unlocking-efficiency-how-ava-became-our-ai-productivity-partner/, [Accessed 25-04-2025].

[34] J. Gomezrangel, A. Vazquez, Y. Lee, K. A. Demir, and B. Liu, "Rising fast, prone to risk: How open-source llm-powered apps are designed and secured," in *Proceedings of the 2025 International Workshop on Artificial Intelligence × Software Engineering (AIxSE)*. IEEE, 2025, accepted for publication; to appear in IEEE Xplore.

[35] "ollama/ollama: Get up and running with Llama 3, Mistral, Gemma, and other large language models. — github.com," https://github.com/ollama/ollama, 2024, [Accessed 31-05-2024].

[36] "ggml-org/llama.cpp: LLM inference in C/C++ — github.com," https://github.com/ggml-org/llama.cpp, [Accessed 16-04-2025].

[37] E. D. Giacinto, "Localai: The free, open source openai alternative," https://github.com/go-skynet/LocalAI, 2023.

[38] "OWASP Top 10 API Security Risks x2013; 2023 - OWASP API Security Top 10 — owasp.org," https://owasp.org/API-Security/editions/2023/en/0x11-t10/, [Accessed 11-04-2024].

[39] "Api security: Latest insights & key trends - 2022 research report," https://services.google.com/fh/files/misc/google_cloud_api_security_research_report.pdf, [Accessed 16-04-2024].

[40] M. Wei, N. S. Harzevili, Y. Huang, J. Yang, J. Wang, and S. Wang, "Demystifying and detecting misuses of deep learning apis," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–12.

[41] "Risk-Assessment-Framework/Raf-Scanner: Raf Scanner IDE," https://github.com/Risk-Assessment-Framework/Raf-Scanner, [Accessed 16-04-2024].

[42] "OWASP/RiskAssessmentFramework: The Secure Coding Framework," https://github.com/OWASP/RiskAssessmentFramework, [Accessed 16-04-2024].

[43] M. Backes, M. Maffei, and K. Pecina, "A security api for distributed social networks." in *Ndss*, vol. 11, 2011, pp. 35–51.

[44] "Detecting misuses of security apis: A systematic review," 2023.

[45] "What is Apigee? : Google Cloud — cloud.google.com," https://cloud.google.com/apigee/docs/api-platform/get-started/what-apigee, [Accessed 16-04-2024].

[46] "Cloud Armor Network Security — cloud.google.com," https://cloud.google.com/security/products/armor?hl=en, [Accessed 16-04-2024].

[47] "reCAPTCHA — cloud.google.com," https://cloud.google.com/security/products/recaptcha?hl=en, [Accessed 16-04-2024].

[48] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, "Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 722–737. [Online]. Available: https://doi.org/10.1145/3620666.3651380

[49] J. Echterhoff, F. Faghri, R. Vemulapalli, T.-Y. Hu, C.-L. Li, O. Tuzel, and H. Pouransari, "Muscle: A model update strategy for compatible llm evolution," *arXiv preprint arXiv:2407.09435*, 2024.

[50] R. Bairi, A. Sonwane, A. Kanade, V. D. C., A. Iyer, S. Parthasarathy, S. Rajamani, B. Ashok, and S. Shet, "Codeplan: Repository-level coding using llms and planning," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, Jul. 2024. [Online]. Available: https://doi.org/10.1145/3643757

[51] N. Nahar, C. Kästner, J. Butler, C. Parnin, T. Zimmermann, and C. Bird, "Beyond the comfort zone: Emerging solutions to overcome challenges in integrating llms into software products," 2024. [Online]. Available: https://arxiv.org/abs/2410.12071

[52] B. Abeysinghe and R. Circi, "The challenges of evaluating llm applications: An analysis of automated, human, and llm-based approaches," 2024. [Online]. Available: https://arxiv.org/abs/2406.03339

[53] C. Deng, Y. Duan, X. Jin, H. Chang, Y. Tian, H. Liu, Y. Wang, K. Gao, H. P. Zou, Y. Jin, Y. Xiao, S. Wu, Z. Xie, W. Lyu, S. He, L. Cheng, H. Wang, and J. Zhuang, "Deconstructing the ethics of large language models from long-standing issues to new-emerging dilemmas: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2406.05392

[54] "PyCQA/bandit: Bandit is a tool designed to find common security issues in Python code. — github.com," https://github.com/PyCQA/bandit, [Accessed 17-07-2025].

[55] "Docs home — Semgrep — semgrep.dev," https://semgrep.dev/docs/, [Accessed 17-07-2025].

[56] "protectai/llm-guard: The Security Toolkit for LLM Interactions — github.com," https://github.com/protectai/llm-guard, [Accessed 17-07-2025].

[57] "google/osv-scanner: Vulnerability scanner written in Go which uses the data provided by https://osv.dev — github.com," https://github.com/google/osv-scanner/, [Accessed 18-07-2025].

[58] "OSV - Open Source Vulnerabilities — osv.dev," https://osv.dev/#use-vulnerability-scanner, 2025, [Accessed 18-07-2025].

[59] "Agents — langchain.com," https://www.langchain.com/agents, [Accessed 24-07-2025].

[60] "open-webui/open-webui: User-friendly AI Interface (Supports Ollama, OpenAI API, ...) — github.com," https://github.com/open-webui/open-webui, [Accessed 23-07-2025].

[61] "ChatGPTNextWeb/NextChat: Light and Fast AI Assistant. Support: Web — iOS — MacOS — Android — Linux — Windows — github.com," https://github.com/ChatGPTNextWeb/NextChat, [Accessed 23-07-2025].

[62] "lobehub/lobe-chat: Lobe Chat - an open-source, modern-design LLMs/AI chat framework. Supports Multi AI Providers(OpenAI / Claude 3 / Gemini / Perplexity / Bedrock / Azure / Mistral / Ollama), Multi-Modals (Vision/TTS) and plugin system. One-click FREE deployment of your private ChatGPT chat application." https://github.com/lobehub/lobe-chat, [Accessed 11-04-2024].

[63] "infiniflow/ragflow: RAGFlow is an open-source RAG (Retrieval-Augmented Generation) engine based on deep document understanding. — github.com," https://github.com/infiniflow/ragflow, [Accessed 23-07-2025].

[64] "Mintplex-Labs/anything-llm: A multi-user ChatGPT for any LLMs and vector database. Unlimited documents, messages, and storage in one privacy-focused app. Now available as a desktop application with a built-in LLM!" https://github.com/Mintplex-Labs/anything-llm?tab=readme-ov-file, [Accessed 04-04-2024].

[65] "mendableai/firecrawl: Turn entire websites into LLM-ready markdown or structured data. Scrape, crawl and extract with a single API. — github.com," https://github.com/mendableai/firecrawl, [Accessed 21-04-2025].

[66] "khoj-ai/khoj: Your AI second brain. Self-hostable. Get answers from the web or your docs. Build custom agents, schedule automations, do deep research. Turn any online or local LLM into your personal, autonomous AI (gpt, claude, gemini, llama, qwen, mistral). — github.com," https://github.com/khoj-ai/khoj, [Accessed 23-07-2025].

[67] "GaiZhenbiao/ChuanhuChatGPT: GUI for ChatGPT API and many LLMs. Supports agents, file-based QA, GPT finetuning and query with web search. All with a neat UI." https://github.com/GaiZhenbiao/ChuanhuChatGPT, [Accessed 12-04-2024].

[68] "arc53/DocsGPT: DocsGPT is an open-source genAI tool that helps users get reliable answers from knowledge source, while avoiding hallucinations. It enables private and reliable information retrieval, with tooling and agentic system capability built in. — github.com," https://github.com/arc53/DocsGPT, [Accessed 23-07-2025].

[69] "onyx-dot-app/onyx: Gen-AI Chat for Teams - Think ChatGPT if it had access to your team's unique knowledge. — github.com," https://github.com/onyx-dot-app/onyx, [Accessed 23-07-2025].

[70] "vllm-project/vllm: A high-throughput and memory-efficient inference and serving engine for LLMs," https://github.com/vllm-project/vllm, [Accessed 16-04-2024].

[71] "aws/sagemaker-python-sdk: A library for training and deploying machine learning models on Amazon SageMaker — github.com," https://github.com/aws/sagemaker-python-sdk, [Accessed 23-07-2025].

[72] "REST API endpoints for search - GitHub Docs — docs.github.com," https://docs.github.com/en/rest/search/search?apiVersion=2022-11-28, [Accessed 11-04-2025].

[73] "NVD: National vulnerability database - Home — nvd.nist.gov," https://nvd.nist.gov, [Accessed 26-09-2025].

[74] M. Ikram, R. Masood, G. Tyson, M. A. Kaafar, N. Loizon, and R. Ensafi, "The chain of implicit trust: An analysis of the web third-party resources loading," in *The World Wide Web Conference*, 2019, pp. 2851–2857.

[75] F. Jiang, Z. Xu, L. Niu, B. Wang, J. Jia, B. Li, and R. Poovendran, "Poster: Identifying and mitigating vulnerabilities in llm-integrated applications," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1949–1951. [Online]. Available: https://doi.org/10.1145/3634737.3659433

[76] F. Li, Z. Durumeric, J. Czyz, M. Karami, M. Bailey, D. McCoy, S. Savage, and V. Paxson, "You've got vulnerability: Exploring effective vulnerability notifications," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1033–1050.

[77] B. Chinthanet, R. G. Kula, S. McIntosh, T. Ishio, A. Ihara, and K. Matsumoto, "Lags in the release, adoption, and propagation of npm vulnerability fixes," *Empirical Softw. Engg.*, vol. 26, no. 3, May 2021. [Online]. Available: https://doi.org/10.1007/s10664-021-09951-x

[78] T. Menzies, W. Nichols, F. Shull, and L. Layman, "Are delayed issues harder to resolve? revisiting cost-to-fix of defects throughout the lifecycle," *Empirical Software Engineering*, vol. 22, no. 4, pp. 1903–1935, 2017.

[79] N. Alexopoulos, M. Brack, J. P. Wagner, T. Grube, and M. Mühlhäuser, "How long do vulnerabilities live in the code? a {Large-Scale} empirical measurement study on {FOSS} vulnerability lifetimes," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 359–376.

[80] J. Kim, W. Choi, and B. Lee, "Prompt flow integrity to prevent privilege escalation in llm agents," *arXiv preprint arXiv:2503.15547*, 2025.

[81] B. Jayaraman, V. J. Marathe, H. Mozaffari, W. F. Shen, and K. Kenthapadi, "Permissioned llms: Enforcing access control in large language models," *arXiv preprint arXiv:2505.22860*, 2025.

[82] Y. Wang, C. Cai, Z. Xiao, and P. E. Lam, "Llm access shield: Domain-specific llm framework for privacy policy compliance," *arXiv preprint arXiv:2505.17145*, 2025.

[83] "Dependabot — github.com," https://github.com/dependabot, [Accessed 24-07-2025].

[84] C. Xiang, Y. Wu, B. Shen, M. Shen, H. Huang, T. Xu, Y. Zhou, C. Moore, X. Jin, and T. Sheng, "Towards continuous access control validation and forensics," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 113–129.

[85] L. Bauer, S. Garriss, and M. K. Reiter, "Detecting and resolving policy misconfigurations in access-control systems," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, pp. 1–28, 2011.

[86] B. Lu, X. Zhang, Z. Ling, Y. Zhang, and Z. Lin, "A measurement study of authentication rate-limiting mechanisms of modern websites," in *Proceedings of the 34th annual computer security applications conference*, 2018, pp. 89–100.

[87] Y. Jiang, F. Fu, X. Yao, G. He, X. Miao, A. Klimovic, B. Cui, B. Yuan, and E. Yoneki, "Demystifying cost-efficiency in llm serving over heterogeneous gpus," 2025. [Online]. Available: https://arxiv.org/abs/2502.00722

[88] "Remote Code Execution Vulnerability in vLLM Multi-Node Cluster Configuration — github.com," https://github.com/vllm-project/vllm/security/advisories/GHSA-9pcc-gvx5-r5wm, [Accessed 24-07-2025].

[89] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 25–35.

[90] X. Tan, Y. Zhang, J. Cao, K. Sun, M. Zhang, and M. Yang, "Understanding the practice of security patch management across multiple branches in oss projects," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 767–777.

[91] S. A. Akhavani, B. Ousat, and A. Kharraz, "Open source, open threats? investigating security challenges in open-source software," *arXiv preprint arXiv:2506.12995*, 2025.

[92] "Android Chat SDK - Kotlin Messaging SDK — getstream.io," https://getstream.io/chat/sdk/android/?utm_source=Github\&utm_medium=Jaewoong_OSS\&utm_content=Developer\&utm_campaign=Github_April2024_Jaewoong_ChatGPT\&utm_term=DevRelOss, [Accessed 30-05-2024].

[93] F. Li and V. Paxson, "A large-scale empirical study of security patches," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2201–2215.

[94] F. Zampetti, S. Geremia, G. Bavota, and M. Di Penta, "Ci/cd pipelines evolution and restructuring: A qualitative and quantitative study," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 471–482.

[95] V. Alto, *Building LLM Powered Applications: Create intelligent apps and agents with large language models*. Packt Publishing Ltd, 2024.

[96] E. Kandogan, N. Bhutani, D. Zhang, R. L. Chen, S. Gurajada, and E. Hruschka, "Orchestrating agents and data for enterprise: A blueprint architecture for compound ai," *arXiv preprint arXiv:2504.08148*, 2025.

[97] D. Quinn, M. Nouri, N. Patel, J. Salihu, A. Salemi, S. Lee, H. Zamani, and M. Alian, "Accelerating retrieval-augmented generation," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2025, pp. 15–32.

[98] K.-T. Tran, D. Dao, M.-D. Nguyen, Q.-V. Pham, B. O'Sullivan, and H. D. Nguyen, "Multi-agent collaboration mechanisms: A survey of llms," *arXiv preprint arXiv:2501.06322*, 2025.

[99] J. Zhou, M. Pacheco, Z. Wan, X. Xia, D. Lo, Y. Wang, and A. E. Hassan, "Finding a needle in a haystack: Automated mining of silent vulnerability fixes," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 705–716.

[100] X. Yang, W. Zhu, M. Pacheco, J. Zhou, S. Wang, X. Hu, and K. Liu, "Code change intention, development artifact, and history vulnerability: Putting them together for vulnerability fix detection by llm," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 489–510, 2025.