Mapping the Landscape of LLM Deployment in the Wild: Prevalence, Patterns, and Perils

XINYI HOU[*‡], Huazhong University of Science and Technology, China
JIAHAO HAN[*‡], Huazhong University of Science and Technology, China
YANJIE ZHAO[‡], Huazhong University of Science and Technology, China
SHENAO WANG[‡], Huazhong University of Science and Technology, China
HAOYU WANG[†‡], Huazhong University of Science and Technology, China

Large language models (LLMs) are increasingly deployed through open-source and commercial frameworks, enabling individuals and organizations to self-host advanced LLM capabilities. As LLM deployments become prevalent, ensuring their secure and reliable operation has become a critical issue. However, insecure defaults and misconfigurations often expose LLM services to the public internet, posing serious security risks. This study conducted a large-scale empirical investigation of public-facing LLM deployments, focusing on the prevalence of services, exposure characteristics, systemic vulnerabilities, and associated risks. Through internet-wide measurements, we identified 320,102 public-facing LLM services across 15 frameworks and extracted 158 unique API endpoints, categorized into 12 groups based on functionality and security risk. Our analysis found that over 40% of endpoints used plain HTTP, and over 210,000 endpoints lacked valid TLS metadata. API exposure was highly inconsistent: some frameworks, such as Ollama, responded to over 35% of unauthenticated API requests, with about 15% leaking model or system information, while other frameworks implemented stricter controls. We observed widespread use of insecure protocols, poor TLS configurations, and unauthenticated access to critical operations. These security risks, such as model leakage, system compromise, and unauthorized access, are pervasive and highlight the need for a secure-by-default framework and stronger deployment practices.

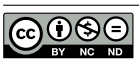
CCS Concepts: • **Networks** → **Network measurement**; • **Security and privacy** → **Distributed systems security**; • **Information systems** → *Service discovery and interfaces.*

Additional Key Words and Phrases: Large Language Models, LLM deployment, Internet Measurement, Security

ACM Reference Format:
Xinyi Hou, Jiahao Han, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2026. Mapping the Landscape of LLM Deployment in the Wild: Prevalence, Patterns, and Perils. *Proc. ACM Meas. Anal. Comput. Syst.* 10, 1, Article 15 (March 2026), 26 pages. https://doi.org/10.1145/3788097

[*]Xinyi Hou and Jiahao Han contributed equally to this work.
[†]Haoyu Wang is the corresponding author (haoyuwang@hust.edu.cn).
[‡]The full name of the authors' affiliation is Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology.

Authors' Contact Information: Xinyi Hou, xinyihou@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, China; Jiahao Han, jiahaohan@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, China; Yanjie Zhao, yanjie_zhao@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, China; Shenao Wang, shenaowang@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, China; Haoyu Wang, haoyuwang@hust.edu.cn, Huazhong University of Science and Technology, Wuhan, China.



ACM 2476-1249/2026/3-ART15
https://doi.org/10.1145/3788097

1 Introduction

Driven by renowned models like OpenAI's GPT-OSS [40] and DeepSeek series [11], large language models (LLMs) are rapidly gaining popularity and reshaping a wide range of tasks, such as natural language understanding [69], software engineering [21], and autonomous systems [55]. **These models, which were once primarily deployed by professionals in industrial settings, are now being increasingly adopted by broader individual developers.** The emergence of user-friendly tools and a variety of community ecosystem, such as Ollama [51], vLLM [54], and ConfyUI [12], has enabled individual developers to deploy and customize powerful LLMs for a variety of tasks, such as content creation [24], software development [21], and investment assistance [60]. However, as LLM deployment becomes increasingly accessible, many implementations lack robust security measures, exposing users and organizations to significant operational and security risks. The OWASP Top 10 for LLM Applications [15] has identified several risks that deserve particular attention during deployment, including sensitive information disclosure, unbounded consumption, and supply chain risks. These issues are particularly pronounced in self-hosted and open-source settings, where models, APIs, and infrastructure are often left vulnerable to public internet exposure without sufficient protection.

Open-source frameworks commonly used for self-hosted LLM deployments often suffer from insecure default settings and misconfigurations [14, 53], **exposing sensitive interfaces to the public internet** without adequate protection and significantly enlarging the attack surface [2]. Several recent security reports [14, 18, 30, 53] have uncovered vulnerabilities and instances of exposed LLM services. For instance, Ollama [51], a framework widely utilized for deploying local LLM services, exposes RESTful APIs publicly by default without authentication [35], enabling unauthorized operations such as model deletion, theft, GPU resource hijacking, and critical remote code execution (e.g., CVE-2024-37032 [36]). Similarly, OpenWebUI [52], commonly integrated alongside Ollama for enhanced interaction capabilities, has suffered from vulnerabilities allowing arbitrary file uploads (CVE-2024-6707 [37]), potentially facilitating remote command execution. Additionally, ComfyUI [12], known for its plugin-rich environment supporting diffusion-based generation tasks, has experienced multiple severe plugin-related security issues [45], including unauthorized remote code execution [10], arbitrary file access [10], and serialization vulnerabilities [3]. These cases make it clear that even widely used LLM frameworks can have serious security issues.

Recent studies have improved the performance and scalability of LLM serving systems [24, 26, 28]. Other researchers have examined security and privacy risks in LLM applications. They have identified threats such as prompt injection, information leakage, and attacks on agent-based or plugin-based systems [19, 20, 42, 57, 58, 64]. Some surveys provide valuable overviews of these security concerns [20, 64]. However, most existing work focuses on isolated technical components or theoretical analysis. Few studies have measured how these risks appear in real-world, internet-facing LLM deployments. Although the types of risks may not be new, their exploitation in the context of publicly exposed LLM services is both highly practical and largely uncharted. Attacks targeting open LLM endpoints can be executed with minimal effort and have direct, real-world consequences, given the unique capabilities and widespread adoption of these systems.

Motivated by these observations, our work aims to fill this gap by systematically investigating insecure practices and vulnerabilities in open LLM deployments. Our investigation is guided by the following research questions (RQs):

RQ1 What is the global distribution and typical setup of public LLM services in the wild, and what are the key challenges in identifying and characterizing them at scale? We aim to map the current landscape of public LLM deployments (e.g., geographic, organizational,

and technical characteristics) and discuss the key challenges and limitations in Internet-wide identification and characterization of public LLM deployments.

RQ2 How do major LLM deployment frameworks differ in API exposure and access control? This helps us understand the diversity of deployment strategies and the effectiveness of default security mechanisms.

RQ3 What are the main security risks and vulnerabilities in public LLM deployments? Building on RQ2, we identify and categorize recurring risk patterns from observed real-world endpoint behaviors and quantify their prevalence across frameworks to assess the most pressing security threats facing public LLM services.

We conduct a large-scale empirical study to examine the prevalence and characteristics of public-facing LLM deployments. We first identified 15 popular LLM deployment frameworks and then systematically discovered public LLM services using an internet asset search engine. This process identified 320,102 unique instances. **To address RQ1,** we analyzed these services by geography, organization, protocol, port, and other factors, finding that over 40% of services used plain HTTP, Notably, 39% of these HTTP services lack any valid domain name and are accessible only by IP address. Over 210,000 services lacked valid TLS metadata. These findings indicate widespread vulnerabilities in transport security. **To answer RQ2,** we collected 158 unique API endpoints from official documentation and categorized them into 12 functional categories. We probed the discovered services to examine how their APIs were exposed. The results revealed significant differences between frameworks. For example, Ollama and Llamafile responded to over 10% of unauthenticated requests. In contrast, Open WebUI and Gradio responded to less than 0.1%. We found that sensitive models or system information were frequently leaked. In Ollama deployments, this percentage exceeded 14%. **For RQ3,** we interpreted the security implications of these exposed behaviors and summarized five recurring risk patterns, including model information disclosure, system configuration disclosure, unauthorized access, vulnerabilities & reverse, and sensitive content generation. We substantiate each risk category with representative real-world endpoint response cases and quantify its prevalence across deployment frameworks. For example, ComfyUI exhibits exposure across all five risk categories, with endpoints such as `/history` and `/object_info` accessible in about 10% of sampled deployments. Based on these findings, we provide actionable security practice recommendations for stakeholders, including framework providers and users. Our primary contributions[1] are as follows:

- We collected and curated the first large-scale dataset of public LLM service deployments, comprising 320,102 unique endpoints across 15 major LLM deployment frameworks. Using this dataset, we systematically analyze the current state of public LLM deployments, including geographic and organizational distribution, server stack composition, domain usage, and communication security. This dataset can also support future research.
- We implemented an automated detection framework tailored for LLM services that scans and identifies exposed API endpoints and assesses the extent of API surface exposure across different deployment frameworks.
- We conducted a detailed security analysis using information disclosed by 158 API endpoints across 12 categories, revealing five common security risks in public LLM deployments, including model information disclosure, system configuration disclosure, unauthorized access, vulnerabilities & reverse, and sensitive content generation.

The remainder of this paper is organized as follows: § 2 introduces common LLM deployment paradigms and tools and provides the motivation for this research. § 3 describes our measurement pipeline. § 4 answers the three RQs, including key findings regarding exposure, deployment patterns,

[1]Our artifacts are publicly available at https://github.com/security-pride/LLM_Deployment.

and security risks. § 5 discusses broader implications, limitations, and potential mitigations. § 6 summarizes related research work. Finally,§ 7 concludes this paper.

2 Background

2.1 LLM Deployment Paradigms and Tooling

The deployment of LLMs has evolved significantly in recent years, giving rise to various paradigms and tools that cater to diverse use cases and user needs. Broadly, LLM deployment can be categorized into cloud-based and self-hosted approaches. Cloud-based deployment is the most common paradigm, where models are hosted and managed by providers such as OpenAI, Google, and AWS, offering scalability, ease of integration, and regular updates. Several cloud service providers have also launched Model-as-a-Service (MaaS) platforms [16], such as OpenAI's GPT API service [39], Volvano Ark [13], and Silicon Flow [7]. In contrast, self-hosted deployment empowers users to run LLMs on their own infrastructure, providing greater control over customization, data privacy, and cost management. This approach has gained traction among small enterprises, research organizations, and individual developers, particularly with the rise of open-source LLMs. To support the self-hosted deployment paradigm, a variety of tools and frameworks have emerged. Drawing on the hierarchical classification outlined in Alpha Lab of Topsec's 2025 LLM Component Security Report [1], we categorize LLM deployment frameworks into four functional categories: inference engines, service frameworks, application user interfaces, and developer tools. As illustrated in Figure 1, these components form a typical LLM deployment workflow.

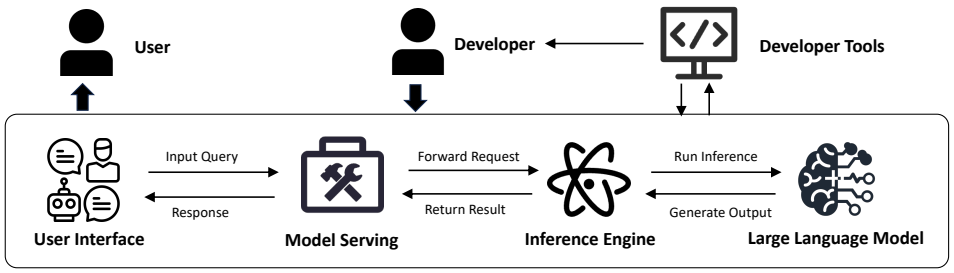


Fig. 1. Paradigms of LLM Deployment.

Inference engines typically serve as the underlying core components of LLM operations, focusing on improving model inference speed, optimizing hardware resource utilization, and supporting large-scale concurrent requests. These engines lay the foundation for LLM service deployment and application development, and are suitable for a variety of high-performance scenarios. Mainstream inference engines include vLLM [54], llama.cpp [17], and Llamafile [43]. **Model serving frameworks**, such as Ray Serve [23] and Ollama [51], serve as a bridge between underlying models and upper-layer applications. These frameworks not only support encapsulating models as standard service interfaces but also implement automated scaling, load balancing, and multi-node collaboration. These capabilities enable developers to more flexibly manage and schedule model services to meet dynamically changing business needs. **Application interfaces** provide user-friendly interfaces and rich integration methods for interacting with models, such as OpenWebUI [52] and Gradio [49]. They significantly lower the barrier to entry for non-expert users to use large models, enabling developers to quickly build demonstration, testing, or prototype systems. **Developer tools** focus on the full lifecycle of model development and integration, covering model training, deployment, evaluation, and integration with external systems. These tools bring standardization

and automation to the development process, helping teams efficiently implement complex model application scenarios while improving the maintainability and reusability of the system.

2.2 Motivation Example

Ollama has become a popular framework for deploying LLMs due to its simplicity and flexibility. However, in its default configuration, all versions of Ollama expose the API port (11434) without mandatory authentication, which poses a risk of unauthorized access (CNVD-2025-04094 [35]). This default configuration means that the LLM service is publicly accessible on the public internet. For example, anyone can access the `/api/tags` endpoint to enumerate all LLMs installed on the server, as shown in Figure 2. Accessing the `/api/chat` endpoint allows direct interaction with deployed models and initiating conversations. More concerning, critical endpoints such as `/api/delete` are accessible without any security checks, allowing attackers to delete models from the server with a single request. This exposure poses a serious security risk: unauthorized users could potentially obtain sensitive information about models in use, abuse the service for large or malicious queries, or even disrupt operations by deleting or manipulating critical resources.


```
{
  "models": [
    {
      "name": "yantien/llama3.1- uncensored:latest",
      "model": "yantien/llama3.1-uncensored:latest",
      "modified_at": "2025-04-
10T11:35:09.851705505+08:00",
      "size": 4661227048,
      "digest":
"e62e31cfe2148287bbf658a036e84e37ce51716da35ed3a0e0
bd5c4e19786db3",
      "details": {
        "parent_model": "",
        "format": "gguf",
        "family": "llama",
        "families": [
          "llama"
        ],
        "parameter_size": "8.0B",
        "quantization_level": "Q4_0"
      }
    },
    {
      "name": "llama3.1:8b",
      "model": "llama3.1:8b",
      "modified_at": "2025-04-
10T09:41:31.873117524+08:00",
      "size": 4920753328,
      "digest":
"46e0c10c039e019119339687c3c1757cc81b9da49709a3b39248
63ba87ca666e",
      "details": {
        "parent_model": "",
        "format": "gguf",
        "family": "llama",
        "families": [
          "llama"
        ],
        "parameter_size": "8.0B",
        "quantization_level": "Q4_K_M"
      }
    }
  ]
}
```


Fig. 2. Example response from the `/api/tags` endpoint, containing detailed information about all installed models, including model names, versions, sizes, modification times, and parameter configurations.

Motivated by the security risks highlighted by the Ollama case, we intend to further explore the current state of publicly exposed LLM services. This includes systematically measuring the number of LLM service endpoints accessible from the open internet and analyzing their deployment characteristics, common exposure patterns, and associated security risks. By understanding the prevalence and nature of these exposed services, we hope to provide a comprehensive picture of the attack surface facing LLM deployments in the real world and highlight areas where improved security practices and mitigations are needed.

3 Methodology

This section details the methodology we use to analyze LLM deployments in the wild. Figure 3 provides an overview of the pipeline.. We first select representative deployment frameworks (§ 3.1), then discover publicly accessible instances (§ 3.2), probe their APIs to collect metadata (§ 3.3), and finally analyze their configurations and security posture (§ 3.4).

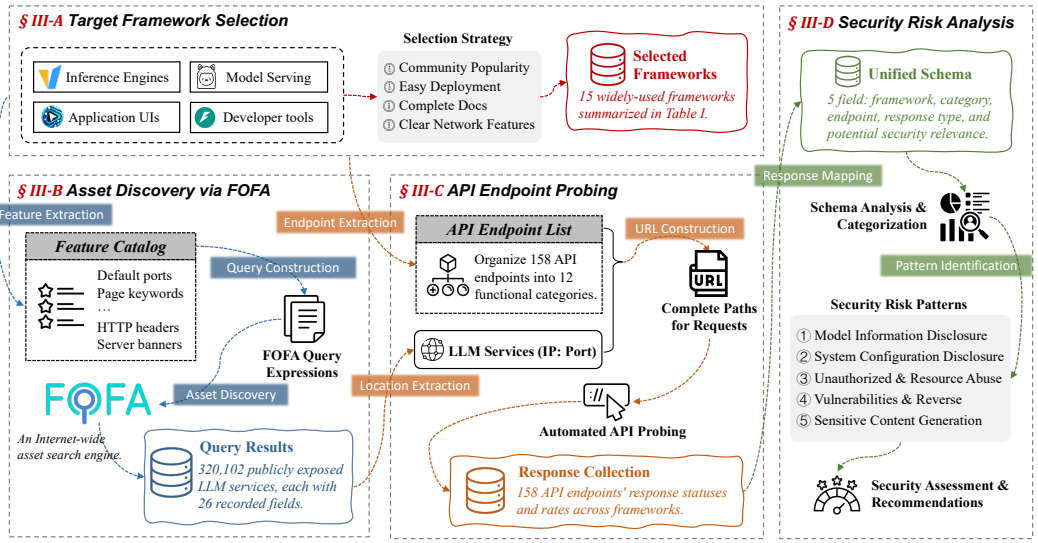


Fig. 3. Overview of the Empirical Analysis Pipeline.

3.1 Target Framework Selection

To structure our measurement and ensure broad coverage across the LLM deployment stack, we selected representative frameworks across four functional categories: **Inference engines** (e.g., vLLM [54], llama.cpp [17]) handle local model execution optimized for performance. **Model serving frameworks** (e.g., Ollama [51], Ray Serve [23]) expose scalable APIs. **Application UIs** (e.g., Open WebUI [52], Jan [50], ComfyUI [12]) provide user-facing interfaces, while **developer tools** (e.g., Jupyter Notebook [25], FastAPI [44]) facilitate development workflows and integration. Our framework selection followed a multi-step process: First, we surveyed the most cited projects in open source communities (e.g., GitHub, Hugging Face) and industry forums to compile an initial shortlist. We then screened these frameworks based on factors such as ease of deployment (e.g., availability of Docker images or installation scripts), quality and completeness of documentation, and evidence of active maintenance (recent commits, issue activity). Next, we prioritized frameworks that exhibited unique network exposure patterns and supported diverse deployment scenarios, such as on-premises, cloud, and hybrid deployments. This process enabled us to capture both widely adopted solutions and emerging tools with unique characteristics. In total, we selected 15 widely used frameworks, which are summarized in Table 1.

3.2 Asset Discovery via FOFA

We use FOFA [8], a widely adopted internet-wide asset search engine that indexes IP addresses, domains, and service metadata [8], to discover public-facing LLM deployments at scale. For each framework, we built a feature catalog that captures its unique network-level characteristics, such as default port (e.g., Ollama's default port is 11434), page title or HTML keywords (e.g., `title=Open WebUI`), HTTP headers (e.g., `Ollama is running`), favicon hashes, and known API paths (see Table 1 for details). Based on this catalog, we designed custom FOFA query expressions using the platform's advanced syntax. For instance, `app=Ollama && port=11434` for Ollama and `title=FastAPI || body=FastAPI && (port=8000 || port=8080)` for FastAPI).

Table 1. Publicly Exposed LLM Deployment Frameworks Discovered via FOFA (As of April 20, 2025).

Category	Framework	Description	Key Feature	Count
Inference Engine	vLLM [54]	High-throughput, memory-optimized inference service	title/vLLM; port=8000	6,077
	llama.cpp [17]	C/C++ quantized inference engine	header/llama.cpp	4,234
	GPT4All [34]	Local GPT model runtime exposing REST endpoint	title/GPT4All; port=5001	2,572
	Llamafile [43]	Portable single-file LLM runtime	body/Llamafile	39
Model Serving	Ollama [51]	Cross-platform CLI for local LLM API service	app/Ollama; port=11434	155,423
	AnythingLLM [48]	Local knowledge base integration with LLM API	title/AnythingLLM; port=3001	3,766
	Ray Serve [23]	Scalable microservice framework with autoscaling	body/Ray Serve; port=8265	365
Application UI	Open WebUI [52]	Ollama/GPT-API Web dashboard	title/Open WebUI; port=8080	37,242
	Jan [50]	Interactive local chat UI	title/Jan; port=1337	28,445
	NextChat [9]	Local ChatGPT-style interface	title/ChatGPT Next Web	25,883
	ComfyUI [12]	Visual workflow builder for LLM and image pipelines	title/ComfyUI; port=8188	15,219
	Gradio [49]	Python toolkit for shareable LLM web demos	title/Gradio; port=7860	9,729
	Text Generation Web UI [38]	Generic LLM Web interface	body/text-generation-webui	2,051
Developer Tools	Jupyter Notebook [25]	Interactive Python notebook environment	title/Jupyter Notebook	24,531
	FastAPI/Swagger UI [44]	Auto-generated API docs and interactive endpoints	title/FastAPI	4,526
Total	/	/	/	**320,102**

We minimized false positives and false negatives through an iterative process of query refinement and manual validation. Initial FOFA queries were developed for each framework based on these network-level features, after which we systematically collected and examined public service samples to evaluate accuracy. This involved checking banners, landing pages, HTTP headers, and API metadata to verify whether the detected services matched the intended LLM framework. When a query returned irrelevant results, often because of generic keywords or ports that appeared in multiple unrelated services, we refined or replaced these patterns to improve precision. After finalizing the optimized queries, we manually validated representative samples from each framework' results. By recording the proportion of correctly identified endpoints, we further adjusted our approach whenever we observed repeated misclassifications. Although a small margin of error may persist, particularly for highly customized or atypical deployments, this methodology proved effective at reducing both types of error. We performed manual sampling validation, yielding an average false positive rate of 2.8%. All FOFA query expressions are provided in our artifact[2].

As of April 20, 2025, we had identified 320,102 publicly accessible LLM-related services across 15 representative deployment tools, as shown in Table 1. These results form the foundation for the subsequent phases of our study, including endpoint probing and configuration analysis.

3.3 API Endpoint Probing

We extracted 158 API endpoints from the official documentation of 15 widely used LLM deployment frameworks. These endpoints span many functionalities, including model management, file access, kernel and session management, and application deployment. As summarized in Table 2, the endpoints are organized into 12 functional categories. For instance, endpoints under the *Text/Chat* category enable OpenAI-style text generation (e.g., `/v1/chat/completions`), while the *Model Control* group provides access to model lifecycle operations such as loading or deleting models (e.g., `/models/delete`). Several endpoints were found to expose potentially sensitive actions. We define an endpoint as sensitive if it enables actions that could compromise the confidentiality, integrity, or availability of the service or its data, such as file uploads, session control, server-side code execution, model management, or system configuration changes. Using the extracted paths, we combined them with the network locations of previously identified LLM services to construct complete URLs.

[2]The full set of FOFA queries is available at https://github.com/security-pride/LLM_Deployment/blob/main/data_collection/FOFA_Query.yaml.

An automated process then probed these URLs with HTTP(S) requests, systematically collecting and storing all responses for further analysis. The resulting dataset provides a comprehensive view of exposed API functionalities and their response behaviors in real-world deployments.

Table 2. API Categories in Exposed LLM Services and Associated Security Risks.**Risk Types:** MID: Model Information Disclosure; SCD: System Configuration Disclosure; URA: Unauthorized & Resource Abuse; VR: Vulnerabilities & Reverse; SCG: Sensitive Content Generation.

Category	Function	#	MID	SCD	URA	VR	SCG
Text and Chat Generation	Generate or complete text and chat conversations	23	✓				✓
Embedding Generation	Generate text embeddings for search or tasks	9	✓				
Image and Audio Processing	Generate or process images and audio	32					✓
Model Operations	Manage models: load, list, delete, or inspect	28	✓		✓	✓	
File Operations	Upload, download, or delete files	19	✓		✓		
Knowledge Base / RAG	Upload or query knowledge bases for RAG	3	✓				
Fine-Tuning Tasks	Create or manage fine-tuning jobs and data	10	✓		✓		
Session and Kernel Management	Manage sessions, kernels, or compute clusters	16			✓		
Task Queue Management	Submit tasks or monitor job queues	3			✓		
System Configuration	Get system status or configuration	6		✓	✓		
Moderation Checking	Perform content moderation or safety checks	5					✓
Application Deployment	Deploy or manage LLM-based applications	4			✓	✓	
Total		158					

3.4 Configuration and Security Analysis

Rather than attempting an exhaustive crawl of all known services, our probing process was strategically designed to identify insecure deployment practices within representative samples from each LLM framework. Given that the number of discovered services varied substantially across frameworks, we adopted *proportional random sampling* independently for each framework to ensure fair coverage. Following previous work [4, 67], we determined the required sample size for each group based on standard statistical principles. We set our sampling parameters to achieve a 95% confidence level and a 5% confidence interval. These thresholds are widely used in empirical studies. For estimating proportions with desired precision and confidence, we used the widely adopted formula for sample size calculation:

$$n = \frac{Z^2 \cdot p(1-p)}{e^2} \tag{1}$$

where n is the sample size, Z is the z-score corresponding to the chosen confidence level ($Z = 1.96$ for 95%), p is the estimated proportion (set to 0.5 for maximum variance, yielding a conservative sample size), and e is the allowed margin of error (here, $e = 0.05$ for a ±5% confidence interval). For instance, when the total population is 22,832, the calculated sample size using this formula is 378.

After collecting the endpoint responses, two authors independently analyzed the returned API response. Each response was mapped to a unified schema comprising five key fields: *deployment framework*, *endpoint category*, *endpoint path*, *response type* (e.g., success, denial, error), and *potential security relevance*. During the analysis, we first reviewed the API endpoint descriptions and proposed an initial set of risk categories to guide annotation (i.e., model information disclosure, configuration

leakage and resource abuse). We then paid attention to the specific types of information exposed by each endpoint. While many cases were clear-cut, there was often subjectivity in judging how a particular piece of information might be exploited and which risk category it fit best. For example, the exposure of model configuration parameters might be classified as either information disclosure or resource abuse, depending on interpretation. At the initial stage, the independent annotations resulted in eight distinct risk categories, as each author sometimes proposed finer-grained or overlapping categories based on their own assessment. Initial inter-annotator agreement was 82.4%, leaving 13 endpoints with disagreements. For cases where consensus could not be reached, a third author was invited to adjudicate, and this process continued until all cases reached full agreement.

Our assessment focused on three key areas: endpoint responsiveness to unauthenticated access, the extent of functional exposure across twelve endpoint categories, and the presence of risky operations such as model management, file access, and code execution. We ultimately identified five major categories of security risks in the LLM deployment framework detailed in Table 2.

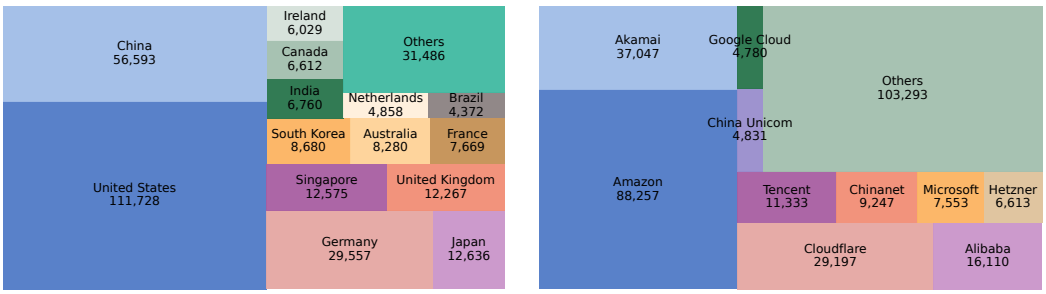
4 Results

This section presents the results of our empirical analysis of public LLM services. Findings are organized by RQs, covering overall exposure, deployment characteristics, configuration patterns, and security risks observed in the dataset.

4.1 RQ1: General Statistics

We begin with basic statistical analyses to summarize the global deployment, organizational involvement, exposure surfaces, and security postures of self-hosted LLM services.

4.1.1 Global and Organizational Deployment Trends. We analyze 320,012 exposed LLM service endpoints, aggregating their origin by country and associated organization. Figure 4 illustrates this landscape with two figures summarizing the top contributors by country and organization.



(a) Geographic Distribution. (b) Organization Distribution.

Fig. 4. Global Landscape of Public LLM Deployment.

Geographic Centralization. As shown in Figure 4a, the United States leads with 111,728 public LLM services, more than double the number of second-place China (56,593). Other prominent contributors include Germany, Japan, and Singapore. This imbalance highlights a pronounced centralization of deployment activity within technologically advanced nations, likely driven by both cloud infrastructure maturity and early AI adoption. Notably, the "Others" category still accounts for over 53,000 instances, suggesting that self-hosted LLMs are also gaining traction in the broader global community, albeit in smaller clusters.

Organizational Dominance and Long Tail. We further analyzed organizational affiliations based on the Autonomous System Numbers (ASNs) of the observed IP addresses. As shown in Figure 4b, a few major providers dominate public LLM deployments. Amazon alone is associated with 88,257 instances, followed by Cloudflare, Akamai, and Microsoft. This concentration is not coincidental: these cloud and CDN providers offer highly accessible and scalable infrastructure, often with free-tier or pay-as-you-go models, making them attractive to both individual developers and small organizations. In many cases, LLM frameworks are also preconfigured for platforms like AWS or Cloudflare Workers, further lowering deployment barriers. Interestingly, a long-tail distribution persists: over 100,000 services are attributed to entities categorized as "Others", which encompass deployments from smaller cloud vendors, local ISPs, academic networks, enterprise networks, hobbyist servers, and edge nodes. Within this category, 13.7% of services lack ASN or organization information. This uneven deployment landscape creates a strategic asymmetry. While **centralized platforms enable efficient patching and policy enforcement, the fragmented long tail poses significant challenges due to inconsistent security practices**. Attackers can exploit this imbalance by targeting the more vulnerable and less secure individually deployed LLM services, while defenders are forced to contend with a broader and more unpredictable attack surface. Notably, in our subsequent analysis of HTTP services, 39% belong to the "Others" category.

4.1.2 High-Traffic Domains and Service Concentration. We found that more than 210,000 exposed LLM services in the FOFA dataset have an empty "domain" field, indicating the absence of valid domain assignments. Without proper DNS or certificate configuration, such services are usually reachable only by IP address and present increased risks as well as management challenges. Beyond these unassigned cases, certain domains are associated with an unusually high number of LLM instances. As shown in Figure 5, domains such as `nellasushi.es`, `mysuccess.be`, and `human-rights-law.eu` each host thousands of services. We note that these domains do not correspond to major LLM providers or recognized organizations. Many of these domains also exhibit short registration histories and lack clear ownership records.

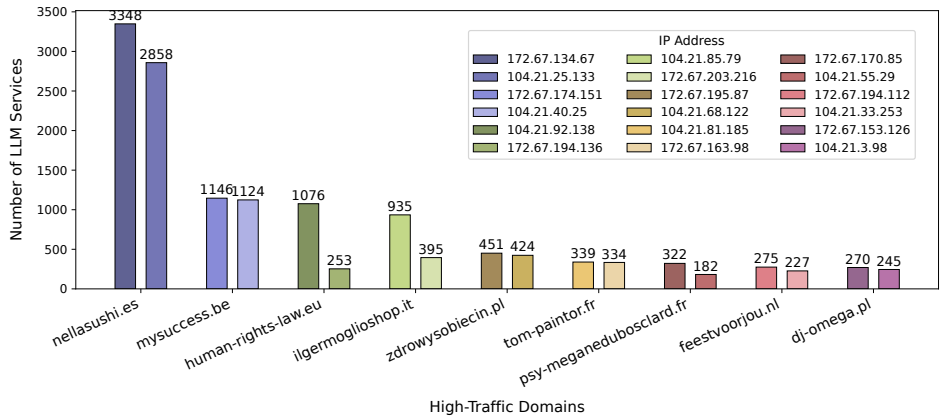


Fig. 5. High-Traffic Domains by Number of LLM Services.

This level of repetition is uncommon in conventional web deployments and suggests that these domains serve as default endpoints in automated or templated hosting environments. Two factors support this interpretation. First, services under the same domain frequently share a small number of IP addresses, indicating centralized hosting or large-scale reuse of identical deployment images.

For instance, 6,206 instances under `nellasushi.es` are served by two IPs. Second, deployment metadata in Figure 6 reveals a consistent reliance on a limited set of frameworks such as ComfyUI, Jan, and vLLM, often in their default configurations. These patterns point to widespread use of prebuilt containers or orchestration scripts. Such concentration has both operational advantages and security implications. Focusing remediation efforts on a few high-frequency domains could reduce exposure at scale. However, **any misconfiguration or compromise within these clusters could simultaneously affect thousands of endpoints**. Moreover, the repeated use of domain names, IPs, and certificates erodes the reliability of trust models and complicates service attribution.

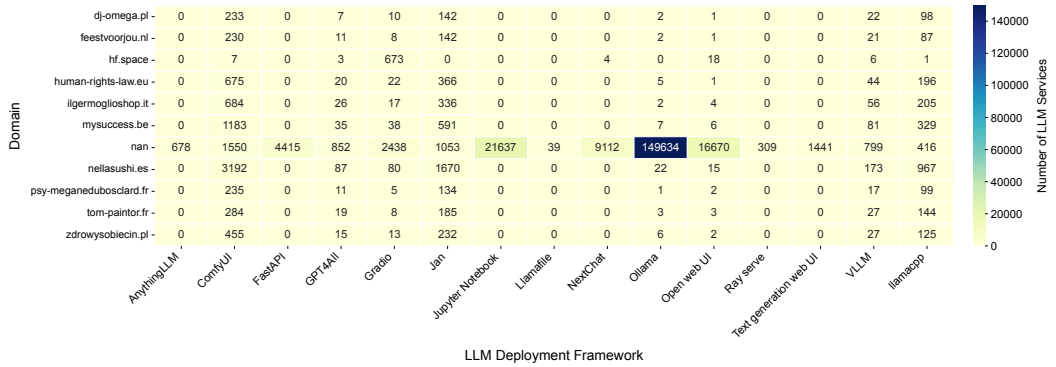


Fig. 6. Distribution of Exposed LLM Services Across Domains and Deployment Frameworks. The heatmap shows that the majority of exposed LLM services lack a valid domain name (nan).

Table 3. Top OS–Server Deployment Combinations in Exposed LLM Services.

OS	Server	Count	Percentage	Share in OS
ubuntu	nginx/1.18.0 (ubuntu)	7,242	42.84%	53.23%
ubuntu	nginx/1.24.0 (ubuntu)	3,659	21.65%	26.9%
windows	microsoft-iis/10.0	567	3.35%	50.04%
ubuntu	apache/2.4.52 (ubuntu)	545	3.22%	4.01%
debian	apache/2.4.62 (debian)	457	2.70%	44.63%
ubuntu	apache/2.4.41 (ubuntu)	420	2.48%	3.09%
ubuntu	nginx/1.14.0 (ubuntu)	389	2.30%	2.86%
ubuntu	apache/2.4.29 (ubuntu)	357	2.11%	2.62%
unix	apache/2.4.62 (unix)	218	1.29%	26.42%
ubuntu	apache/2.4.58 (ubuntu)	194	1.15%	1.43%
ubuntu	nginx/1.26.0 (ubuntu)	194	1.15%	1.43%
debian	apache/2.4.25 (debian)	139	0.82%	13.57%
unix	apache/2.4.57 (unix)	122	0.72%	14.79%
windows	microsoft-iis/8.5	119	0.70%	10.5%
unix	apache/2.4.63 (unix)	117	0.69%	14.18%
ubuntu	nginx/1.10.3 (ubuntu)	112	0.66%	0.82%
windows	microsoft-iis/7.5	101	0.60%	8.91%

4.1.3 Server Stack Composition. Public LLM services are predominantly hosted using familiar and widely adopted server stacks. As shown in Table 3, the most common configuration is `Ubuntu + nginx`, with versions such as `nginx/1.18.0` and `1.24.0` being particularly prevalent. Apache-based deployments are also observed, primarily on Debian or generic Unix systems, while Microsoft IIS

accounts for only a small fraction, reflecting a strong bias toward Linux-based environments. However, a substantial number of services lack complete environment metadata. Over 50,000 instances do not report a valid operating system, and more than 90,000 specify only generic server identifiers. These incomplete configurations likely arise from containerized or scripted deployments where base image information is obscured. While such approaches streamline deployment, they also reduce observability and hinder vulnerability assessment. Some configurations also raise potential security concerns. Instances running outdated server versions (e.g., `nginx/1.14.0`, `apache/2.4.29`) may expose known vulnerabilities if not properly patched [32, 33].

4.1.4 Communication Security and Port Exposure. The communication security of public LLM services varies significantly across ports and deployment frameworks. As shown in Figure 7, a large fraction of services either lack TLS entirely or use outdated versions. Port 443, typically associated with HTTPS, is the most secure, with over 13,000 instances using TLS 1.3 and only 2 instances using TLS 1.0. However, this is not the norm across other ports. Overall, **nearly 129,811 services are still accessible via plain HTTP, accounting for over 40% of the measured endpoints**. This reflects a broad lack of transport encryption across the LLM service ecosystem. Notably, a large number of services on ports such as 11434, 3000, and 8888 lack TLS support. The finding is based on FOFA's active scan results, which record the protocol support observed across the scanned services. These ports are commonly associated with frameworks including Open Web UI, Jan, and Ollama, as shown in Figure 9. For instance, over 32,000 services on port 11434 and nearly 14,000 on 8080 lack any TLS configuration. Even when TLS is present, a non-trivial portion still relies on deprecated versions such as TLS 1.0 and 1.2, which expose services to downgrade attacks and known vulnerabilities.

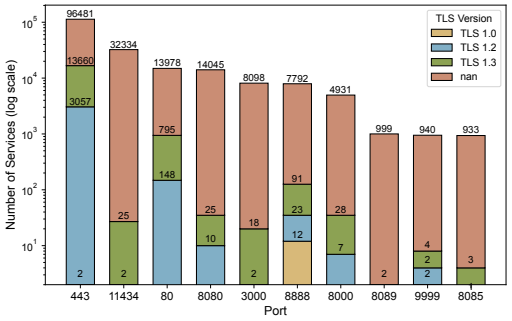


Fig. 7. TLS Version Distribution Across Ports.

Fig. 8. Reused TLS Certificate Subject CNs.

The widespread use of non-standard ports, often with weak or missing encryption, highlights a broader issue: many LLM frameworks prioritize ease of deployment over secure defaults. This creates a communication surface that is both predictable and exposed. Attackers can scan for specific ports tied to popular frameworks and exploit weak encryption to intercept or tamper with LLM outputs. These findings highlight the need for stronger default security in deployment tools, especially in TLS enforcement and port hardening.

4.1.5 Certificate Reuse and Identity Management. TLS certificate metadata associated with public LLM services reveals widespread reuse and misconfiguration, especially in certificate subject fields. As illustrated in Figure 8, common names (CNs) such as `localhost` and domains (e.g., `nellasushi.es`, `mysuccess.be`) appear repeatedly, while a significant number of entries are simply marked as `nan`, indicating missing or unparsable certificate subject fields. Quantitative analysis

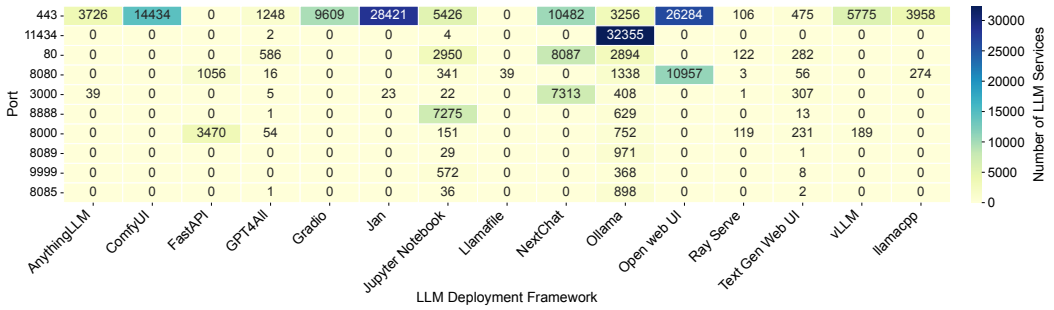


Fig. 9. Distribution of LLM Services Across Ports by Deployment Framework. Darker colors indicate a higher number of exposed LLM service instances on the corresponding port and framework, with prominent concentrations on non-standard and typically unencrypted ports, i.e., 11434 (Ollama).

supports this observation. Over 210,000 services use `localhost` or `nan` as the certificate subject CN, and many others share identical organization names across hundreds or thousands of instances. For certificates with "localhost" as the CN, this is likely indicative of deployment practices where users expect services to be accessible only on non-public or local interfaces. However, such services are frequently exposed to the public Internet due to misconfiguration or misunderstanding of network boundaries. While some reuse is expected in containerized or replicated environments, the scale observed here suggests a lack of certificate management hygiene.

Issuer fields show strong centralization, with certificates mostly issued by `Cloudflare`, `Let's Encrypt`, and `ZeroSSL`, often via automated pipelines. However, they are often paired with missing or generic subject metadata, weakening endpoint authenticity. TLS fingerprinting analysis via `ja3s` values reveals limited diversity, with a few signatures covering most services. This suggests many deployments use default TLS configurations, making them more vulnerable to fingerprint-based traffic correlation or blocking. These patterns expose a weak identity layer in the self-hosted LLM ecosystem, undermining encryption's role in both confidentiality and endpoint authentication.

Answer to RQ1: Our measurement shows that public LLM deployments are highly centralized but security practices remain inconsistent. Of 320,012 endpoints, over half are in the US and China. More than 210,000 services do not have valid domain names; this suggests widespread use of automated or ad-hoc deployments. Thousands of instances reuse domains such as `nellasushi.es`, yet these domains do not correspond to any recognized LLM provider, which raises security concerns due to the difficulty of attribution and the potential for large-scale abuse. Ollama and Open Web UI together account for over 40,000 instances on non-standard ports; nearly all lack TLS. This reflects insecure defaults. Over 210,000 endpoints reuse certificates with generic or missing subject fields, and most certificates originate from only a few issuers. These patterns reveal widespread insecure configurations and highlight how missing or inconsistent metadata makes reliable large-scale identification challenging.

4.2 RQ2: API Exposure

Building on the exposure landscape, we examine API responsiveness across frameworks, coverage of functional categories, and the structure of per-endpoint results.

4.2.1 Framework-Level API Responsiveness. We observed significant variation in API responsiveness among publicly available LLM deployments. As shown in Table 4, frameworks such as Ollama and Llamafile responded to over 10% of unauthenticated API requests, indicating permissive defaults and minimal access controls. In contrast, platforms such as Open WebUI, NextChat, and Gradio had response rates below 0.1%, suggesting more stringent backend protection. FastAPI and AnythingLLM were excluded due to their lack of standardized and comparable APIs.

Table 4. Responsiveness Rate of Major Deployment Frameworks. Proportion of exposed service instances responding to probes, by framework. "Resp. Rate" is the percentage of endpoints giving any response, including or excluding template responses as indicated. (w/): With template response. (w/o): Without template response.

Framework	Ollama			Open WebUI	Jan	Next-Chat	Jupyter Notebook	ComfyUI UI	Gradio	vLLM	llama.-cpp	GPT4-All	Text Gen Web UI	Ray Serve	Llama-file
# Resp.	309(w/)	134(w/o)	22,831	7	523	18	1,626	871	6	19	319	81	6	4	4
Sample/Total	877	877	155,423	37,242	28,445	25,883	24,531	15,219	9,729	6,077	4,234	2,572	2,051	365	39
Resp. Rate	35.23%	15.28%	14.69%	0.02%	1.84%	0.07%	6.63%	5.72%	0.06%	0.32%	7.53%	3.14%	0.31%	1.06%	10.19%

We initially downsampled Ollama for probing because it accounted for over half of all detected endpoints, raising operational and ethical concerns about large-scale scanning. We conducted API probing on a small random sample of 877 instances and allowed us to assess response patterns with minimal impact. After confirming the feasibility and safety of probing, we subsequently performed full probing across all 155,423 Ollama instances for a comprehensive analysis. As shown in the second column of Table 4 , 35.23% of Ollama instances responded. However, upon closer inspection, we discovered that a significant portion of these responses consisted of cookie-cutter "template responses" (generic, uninformative replies) rather than meaningful API output. We speculate that these template responses may be due to default configurations, placeholder endpoints, or defensive mechanisms designed to prevent automated probing. We analyzed template response patterns across different ports, combining file size and content heuristics, and applied automated scripts to systematically exclude all template replies from our dataset. Excluding template responses reduced the actual success rate to 15.28%, closely matching the overall response rate (14.69%) observed in the full Ollama dataset of 155,423 instances. However, we did not observe such template response patterns in other frameworks.

The returned HTTP status codes also reflected security: some services returned 200 (full access), while many returned 401, 403, or 404, indicating authentication, access control, or endpoint obfuscation. Gradio-based deployments often returned 400, reflecting that the API was not designed for direct programmatic use. These response patterns reveal implicit security mechanisms across frameworks. High response rates are often associated with frameworks that expose APIs by default and lack built-in protection, while lower response rates generally correspond to more defensive configurations. Many services respond with structured rejections (401/403), which suggests that some frameworks implement basic security measures even when deployed in public environments.

4.2.2 Functional Coverage of Exposed Endpoints. The functionality exposed through open API endpoints reveals not only the intended capabilities of LLM frameworks but also the extent to which internal operations are externally accessible, whether intentionally or not. As shown in Table 5, the degree of exposure varies substantially across both frameworks and functionality categories. Some frameworks expose broad functionality: for instance, Jan and llama.cpp each respond to over 10 distinct endpoints across text generation, embedding, file operations, and fine-tuning. ComfyUI returns valid responses for 11 categories, including session and queue management, reflecting its interactive, stateful design. In contrast, frameworks like OpenWebUI and Text Generation WebUI

expose few usable endpoints despite implementing many, suggesting stricter access controls or incomplete external integration.

Table 5. API Functionality Coverage Across LLM Deployment Frameworks. Each cell shows "successful / total" API endpoints in that category for the given framework. For example, "2 / 2" in the **Text/Chat Gen** column for Jan means that Jan provides 2 API endpoints in this category, and during our tests, both endpoints had at least one instance where a successful response was observed.

Framework	Text/Chat Gen	Embedding Gen	Image/ Audio	Model Ops	File Ops	Knowleage RAG	Fine-tune	Session Kernel	Task Queue	System Config	Moderation Checking	App Deploy
Ollama	2 / 2	1 / 1		8 / 8						1 / 1		
OpenWebUI	0 / 2	0 / 1		1 / 2	0 / 1	0 / 3						
Jan	2 / 2	1 / 1	6 / 6	2 / 2	2 / 2		2 / 2				1 / 1	
NextChat	0 / 2	0 / 1	0 / 6	1 / 2	0 / 2		0 / 2				0 / 1	
Jupyter Notebook					0 / 6			0 / 13		1 / 1		
ComfyUI		1 / 1	1 / 2	2 / 2	1 / 2		1 / 2	1 / 1	1 / 1		1 / 1	
Gradio	0 / 1				0 / 1				0 / 2	1 / 2		
vLLM	1 / 2	1 / 1	0 / 6	0 / 2	2 / 2		1 / 2				0 / 1	
llama.cpp	4 / 5	1 / 1		1 / 2						1 / 1		
GPT4All	2 / 2			1 / 2								
Text Gen WebUI	2 / 3	1 / 1	6 / 6	3 / 4	2 / 2		1 / 2				1 / 1	
Ray Serve												1 / 3
Llamafile	0 / 2	0 / 1	0 / 6	1 / 2	0 / 2		0 / 2				0 / 1	

Across frameworks, the most commonly exposed functionality is text and chat generation, observed in 11 out of 13 frameworks. Model operations (e.g., listing and loading models) are also frequently reachable in 8 frameworks. However, sensitive features like fine-tuning, moderation, and knowledge base querying remain rare; for instance, only Jan and vLLM expose fine-tuning endpoints, and OpenWebUI supports RAG but with no responsive endpoints. Exposure patterns highlight the trade-off between usability and security. Frameworks designed for local or development use often default to exposing internal APIs, lacking authentication or isolation, while production systems enforce stricter controls. Even non-critical endpoints, such as those for queue status or configuration, can inadvertently leak internal state.

4.2.3 Per-Endpoint Result Representation. All API responses are normalized into a unified schema encompassing *deployment framework*, *endpoint category*, *endpoint path*, *response type*, and *potential security relevance*, enabling consistent analysis across different frameworks. Based on this schema, we analyze Ollama as a representative case study. Table 6 summarizes the endpoint responses for 155,424 observed calls.

Text and chat completion endpoints responded in only 0.19% of cases, indicating limited exposure based on runtime state or configuration, but still potentially revealing system behavior. The embedding endpoint was accessible in 0.22% of attempts; while rarely exposed, its exposure still poses a risk because embeddings can be used in inference attacks. **Model management** endpoints showed uneven availability. Operations such as showing model information and listing running models were most frequently possible (9.94%, 14.40%), while listing local models was moderately possible (4.50%), likely due to its role in coordination. Pull and push operations were low available (0.20%, 0.19%), while destructive actions like delete, copy, and create responded in fewer than 2% of cases, suggesting partial lockdowns, though enforcement remains inconsistent. **System-level** endpoints, including version and runtime model queries, responded in 14.48% of cases, the same level to list running models. The results showed that in Ollama deployments, many endpoints were accessible without authentication, allowing unauthorized users to retrieve sensitive information and exposing serious gaps in access control.

Table 6. Responsiveness of Ollama API Endpoints. Number and response rate (Resp. Rate) of Ollama API endpoint invocations by function category, based on 155,423 service requests.

Category	Endpoint	Function	# Resp.	Resp. Rate
Text/Chat Gen	/api/generate	Generates text completions based on input prompts.	299	0.19%
	/api/chat	Provides multi-turn chat completions for conversational input.	300	0.19%
Embedding	/api/embeddings	Returns vector embeddings for supplied text inputs.	343	0.22%
Model Ops	/api/tags	Lists all models available in the local model repository.	7,001	4.50%
	/api/pull	Downloads a model from a remote registry to local storage.	312	0.20%
	/api/push	Uploads a local model to a remote registry for sharing.	299	0.19%
	/api/delete	Permanently removes a specified model from local storage.	306	0.20%
	/api/copy	Duplicates an existing model under a new name or tag.	1,986	1.28%
	/api/show	Displays detailed information and metadata for a given model.	15,452	9.94%
	/api/create	Creates a new model from provided configuration or data.	311	0.20%
	/api/ps	Lists all currently running or active models in the system.	22,375	14.40%
	/api/running	Shows the models currently loaded and available for inference.	0	0.00%
System Config	/api/version	Returns the current Ollama software version.	22,503	14.48%

Answer to RQ2: API exposure and access control vary across LLM frameworks. Ollama and Llamafile responded to over 10% of unauthenticated requests, while other frameworks, such as Open WebUI and Gradio, responded at rates below 0.1%. Most frameworks exposed basic text generation functionality, but some also leaked model or system information, particularly Ollama, where this rate exceeded 14%. Many frameworks relied on permissive defaults, exposing deployments to the risk of unauthorized access.

4.3 RQ3: Security and Risk Analysis

This section systematically analyzes security risks in LLM deployment frameworks. As shown in Table 7, different frameworks present distinct exposure rates across five risk categories: *model information disclosure*, *hardware and system configuration disclosure*, *unauthorized resource abuse*, *vulnerabilities and reverse engineering*, and *sensitive content generation*. For instance, ComfyUI and Llamafile show relatively high risks in several categories, while frameworks like Jupyter Notebook and Text Gen WebUI display more limited or focused risks. To illustrate our analysis process for each framework, Table 8 provides a concrete example. It lists specific ComfyUI API endpoints associated with each risk type, as well as the number and rate of confirmed cases. The following sections introduce each type of security risk in detail.

4.3.1 Model Information Disclosure. Model information disclosure is among the most prevalent security risks across LLM deployment frameworks, as indicated in Table 7. For example, the `/show` endpoint can reveal detailed model metadata, while the `/history` (Figure 10a) endpoint may leak prior inference workflows, outputs, and fine-tuning traces, compromising user privacy and exposing proprietary data. Notably, in Ollama, the `/api/show` endpoint exhibits an exceptionally high exposure rate of 14.40%, highlighting the severity of this risk even in widely adopted deployment platforms. As shown in Figure 10b, the `/embeddings` endpoint (5.26% exposure) in ComfyUI further illustrates this problem by exposing the list of loaded textual inversion embeddings, such as *Bad-Hands-XL-Embedding_V1.1*, inadvertently revealing deployment purposes. Similarly, in `llama.cpp`, the `/v1/models` (Figure 10c) endpoint may disclose detailed model information, including the actual deployment paths of `.gguf` model files, significantly amplifying the risk of targeted attacks. Such leakage not only facilitates customized attacks but also heightens the risks of model exploitation,

Table 7. Risk Categorization (Min ~ Max Percentage) of Different LLM Deployment Frameworks. Each cell shows the minimum and maximum percentage of instances in a framework exposing any API endpoint in the given risk category (when multiple endpoints exist per risk). If minimum equals maximum, only a single value is shown. "–" indicates no relevant risk observed. Detailed case rates for ComfyUI are summarized in Table 8.

Framework	Model Information Disclosure	System Configuration Disclosure	Unauthorized & Resource Abuse	Vulnerabilities & Reverse	Sensitive Content Generation
ComfyUI	5.26% ~ 10.54%	5.64%	5.06% ~ 5.26%	5.35% ~ 10.09%	5.06%
Ollama	0.00% ~ 14.48%	–	0.19% ~ 1.28%	–	0.19% ~ 0.22%
Text Gen WebUI	0.29%	0.29%	–	–	0.29%
GPT4All	2.87%	–	5.10%	–	–
Llamacpp	3.36% ~ 6.99%	6.71%	11.92%	–	–
Llamafile	8.98% ~ 10.19%	–	9.47%	–	–
Jupyter Notebook	–	8.06%	–	0.16%	–

Table 8. Security Risk Exposure of ComfyUI Endpoints. Percentages ("Risk Rate") represent the proportion of ComfyUI instances (out of 15,219 total) whose corresponding API endpoints responded with confirmed security-relevant content for each risk category. For example, a case rate of 5.29% for /embeddings means that 5.29% of all scanned ComfyUI instances returned actual embedding information from the /embeddings endpoint, thereby exposing this specific risk.

Security Risk	API Endpoint	Description	Risk Rate
Model Information Disclosure	`/embeddings`	Lists all loaded embedding model names.	5.29%
	`/history`	Shows workflow execution history and details.	10.54%
System Configuration Disclosure	`/system_stats`	Reveals server hardware and resource usage.	5.64%
Unauthorized & Resource Abuse	`/queue`	Displays current job queue status.	5.26%
	`/prompt`	Submits or previews inference prompts.	5.06%
Vulnerabilities & Reverse	`/object_info`	Shows metadata about workflow nodes.	10.09%
	`/extensions`	Lists installed extensions and plugins.	5.35%
Sensitive Content Generation	`/prompt`	Allows arbitrary prompt submission.	5.06%

prompt injection, and unauthorized access. In severe cases, attackers could reconstruct user intents, manipulate outputs, or compromise the deployment's integrity and confidentiality.

4.3.2 Hardware and System Configuration Disclosure. Disclosure of system configuration details further enlarges the attack surface by providing adversaries with intelligence about the underlying environment. ComfyUI's `/system_stats` endpoint (5.64%) reveals operating system types, total and available memory, GPU specifications (e.g., RTX 4090 with 24GB VRAM), and ComfyUI version information, as illustrated in Figure 11. Such system insights enable attackers to craft hardware-specific resource exhaustion attacks, such as GPU memory flooding, or identify platform-specific vulnerabilities (e.g., Windows NT kernel exploits). Alarmingly, among instances exposing `/system/stats`, 41.28% were found to be publicly accessible via `–listen 0.0.0.0` without any authentication mechanisms. When combined with the presence of high-performance GPUs, such exposures create ideal conditions for unauthorized resource exploitation, including GPU hijacking, cryptocurrency mining, large-scale model inference, or persistent backdoor implantation.

4.3.3 Unauthorized Access and Resource Abuse. Unauthorized access and resource abuse represent direct threats to the availability and stability of deployed services. Endpoints such as `/queue` (5.26%) and `/prompt` (5.06%) allow unauthenticated users to submit inference tasks or monitor system load. Figure 12 is an example. The ability to observe task queues enables attackers to perform real-time load sensing, identifying idle periods when resource-draining or prompt injection attacks

```
{
  // 1. Malicious payload: reverse shell code embedded in workflow parameters
  "expression": "import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\"185.189.149.151\",8888));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);p=subprocess.call([\"/bin/sh\",\"-i\"]);",

  // 2. Node type and parameters: reveals service logic and node configuration
  "class_type": "EvaluateMultiple1",
  "inputs": {
  "filename_prefix": "ComfyUI",
...
},

  // 3. Error message: exposes internal exceptions and potential vulnerabilities
  "exception_message": "'NoneType' object has no attribute 'decode'",

 // 4. Traceback: reveals server-side file paths and code structure
  "traceback": [
  " File \"/root/www/ComfyUI/execution.py\", line 317,   in execute ...",
...
  ],

  // 5. Workflow structure: exposes internal business logic, node and link details
  "workflow": {
    "nodes": [
      {"id": 2, "type": "VAEDecodeSave", ...},
      {"id": 1, "type": "StringAsAny", ...},
      {"id": 3, "type": "EvaluateMultiple1", ...}
    ],
    "links": [
      ...
    ]
  },

  // 6. Client ID: unique identifier, may be used for tracking or replay attacks
  "client_id": "c43647ea5873403bb28d864fa76f984a"}
```

(a) ComfyUI `/history`. The endpoint exposes workflow parameters, node configurations, error messages, file paths, workflow structure, and client identifiers, revealing service logic and internal implementation details.

```
[
  "CyberRealisticPony_POSV1",
  "CyberRealistic_Negative_PONY-neg",
  "zPDXL3"
  "Bad-Hands-XL-Embedding_V1.1",
  "EasyNegative",
  "EasyNegativeV2_V2.0",
  "Embeddings_Base",
  "FastNegativeV2_V2.0",
  "FastNegativeV2/v2.0/FastNegativeV2",
  "MajicNegative_V2",
  "bad_arien_v1.0",
  "badhandv4",
  "badhandv4-AnimeIllustDiffusion/badhandv4/badhandv4",
  "negative_hand-neg",
  "ng_deepnegative_v1_75t",
  "真实感Negative Embedding for Realistic Vision v2.0_v1.0"
]
```

(b) ComfyUI /embeddings. The endpoint exposes a list of all loaded embedding model names.

```
{
  "object": "list",
  "data": [
    {
      "id": "/home/llama3/Llama3-8B-Chinese-Chat.q5_k_m.GGUF",
      "object": "model",
      "created": 1745728364,
      "owned_by": "llamacpp",
      "meta": {
      "vocab_type": 2,
      "n_vocab": 128256,
      "n_ctx_train": 8192,
      "n_embd": 4096,
      "n_params": 8030261248,
      "size": 5725151232
      }
    }
  ]
}
```

(c) Llama.cpp v1/models. This API call reveals loaded model file paths and configuration details.

Fig. 10. Real-world Examples in Model Information Disclosure.

can be launched with minimal resistance. Furthermore, submitting crafted prompt graphs to `/prompt` without authentication exacerbates the risk: attackers could overload the system with computationally expensive tasks, rapidly depleting GPU memory and causing service outages (denial-of-service attacks). By continuously monitoring queue states, attackers can also infer operational patterns over time, improving the precision and persistence of their abuse strategies.

4.3.4 Vulnerabilities and Reverse Engineering. The exposure of internal modules and metadata creates direct opportunities for vulnerability discovery and reverse engineering. In platforms like ComfyUI, endpoints such as `/extensions` (5.35%) and `/object_info` (10.09%, see an example in Figure 13) reveal detailed information about installed nodes, system structure, and behaviors. With such insights, attackers can reconstruct internal workflows, pinpoint critical components like `PythonEvalNode`, and exploit weaknesses such as insecure APIs or insufficient validation. Nodes handling external communication (e.g., `GeminiAPINode`) may leak API keys, while backend

```
{
  "system": {
    "os": "posix",
    "ram_total": 135063240704,
    "ram_free": 126703878144,
    "comfyui_version": "unknown",
    "python_version": "3.10.15 | packaged by conda-forge | (main, Sep 20 2024, 16:37:05) [GCC 13.3.0]",
    "pytorch_version": "2.4.1+cu121",
    "embedded_python": false,
    "argv": [
      "/cache1/ComfyUI/main.py",
      "--port",
      "5001",
      "--output-directory",
      "/cache1/ComfyUI/output",
      "--input-directory",
      "/cache1/ComfyUI/input",
      "--listen",
      "0.0.0.0",
      "--cuda-device",
      "0"
    ]
  },
  "devices": [
    {
      "name": "cuda:0 NVIDIA GeForce RTX 4090 : cudaMallocAsync",
      "type": "cuda",
      "index": 0,
      "vram_total": 25393692672,
      "vram_free": 24572320508,
      "torch_vram_total": 402653184,
      "torch_vram_free": 12770044
    }
  ]}
```


Fig. 11. A Real-world Example in Hardware and System Configuration Disclosure. ComfyUI `/system/status` endpoint reveals OS details, environment variables, runtime parameters, and device information.


```
{
  "queue_running": [
    [
      14,
      "5cb5c0c7-c15b-4b8d-8c7d-2ea1cb7c0b8a",
      {
        "prompt": {
          "load_image": {
            "class_type": "LoadImage",
            "inputs": {
              "filename": "my_cat.png"
            }
          },
          "detect_cat": {
            "class_type": "DetectCat",
            "inputs": {
              "image": ["@load_image", 0]
            }}}},
      {
        "client_id": "b7e4e1a1-4f0c-4c11-b1ca-8cdfd9aa0f60",
        "submitted_by": "admin"},
      ["detect_cat"]]],
  "queue_pending": [
    [
      15,
      "e0e8a44f-7e2b-45ca-8b41-3f3e4eab8ee2",
      {
        "prompt": {
          "node_a": {
            "class_type": "LoadImage",
            "inputs": {
              "filename": "dog.jpg"
            }
          }
        }
      },
      {
        "client_id": "e6d1e28c-44f8-45e7-97e7-7d9e6c4899f1",
        "submitted_by": "alice"
      },
      ["node_a"]
    ]
  ]
}
```


Fig. 12. A Real-world Example of Unauthorized Access and Resource Abuse. ComfyUI `/queue` endpoint reveals detailed information about both running and pending tasks.

modules (e.g., `cm-api.js`) expose surfaces for command injection. Similar risks arise in another framework. Jupyter Notebook instances exposing the `/api` endpoint disclose version data (e.g., "5.5.0"), allowing attackers to link targets to known vulnerabilities, including unauthorized API access, unauthenticated WebSocket RCE, and token bypass. Fingerprinting deployments and mapping them to public exploits significantly accelerates attack development.

4.3.5 Sensitive Content Generation. Sensitive content generation presents a significant risk, particularly in deployments lacking robust input validation and output moderation. Platforms such as Ollama and Text Generation WebUI expose endpoints that accept custom prompts or generation parameters, which attackers can exploit to induce the production of offensive, illegal, or otherwise sensitive outputs. In our scans, we identified numerous Ollama instances exposing models explicitly labeled as "uncensored" in their model names (e.g., `Qwen2.5-14B_Uncensored_Instruct-Q4_K_M`), reflecting the intentional deployment of models with minimal filtering. Similarly, in Text Generation WebUI, insufficient prompt sanitization may allow adversarial inputs to elicit harmful responses or extract proprietary system behaviors. Although we did not systematically probe the full range of

```
{
  // 1. Model file exposure: reveals available checkpoint/model names
  "ckpt_name": [
    "Adam-Doll.XL | 玩偶盲盒 | 3D手办_V2.safetensors"
  ],

  // 2. Image file exposure: lists internal image files and paths
  "image": [
    "D:\\boy_1.jpg",
    "comfy_20250425161249_2102112902.jpeg",
    "example.png"
  ],

  // 3. Node types and workflow logic: exposes available modules and their parameters
  "KSampler": {
    "input": {
      "required": {
        "model": "MODEL",
        "seed": "INT",
        "steps": "INT",
        "cfg": "FLOAT",
        "sampler_name": [
          "euler", "heun", "dpm_2", "lms"
        ]
      }
  },
    "output": [
      "LATENT"
    ],
    "description": "Uses the provided model, positive and negative conditioning to denoise the latent image."
  },

  // 4. Parameter constraints: reveals valid ranges, defaults and tooltips
  "steps": {
    "type": "INT",
    "default": 20,
    "min": 1,
    "max": 10000,
    "tooltip": "The number of steps used in the denoising process."
  },

  // 5. Implementation information: module names, categories, and descriptions
  "python_module": "nodes",
  "category": "sampling",

  // 6. Internal API references: discloses internal endpoints and routes
  "remote": {
  "route": "/internal/files/output"
  }}
```

Fig. 13. A Real-world Example of Vulnerabilities and Reverse Engineering. ComfyUI `/object_info` endpoint exposes model and image file names, node types, workflow logic, parameter constraints, module descriptions, and internal API routes.

possible outputs for legal and ethical reasons, the public exposure of these endpoints itself constitutes a clear risk vector. If endpoints expose prompt histories, or interaction logs without proper protection, attackers may access sensitive information and further refine their misuse strategies.

Answer to RQ3: Security risk analysis shows that model information disclosure, system leaks, unauthorized access, vulnerabilities, and sensitive content generation are prevalent across LLM deployment frameworks, though unevenly. Frameworks like ComfyUI expose endpoints across all major risk categories, indicating broad and systemic weaknesses. The persistence of such exposures indicates that insecure deployment practices are widespread, and securing LLM systems demands rethinking default configurations, strengthening access controls, and minimizing exposure surfaces beyond patching individual flaws.

5 Discussion

5.1 Recommendations and Best Practices

Improving the security posture of LLM deployments requires coordinated efforts across tooling, usage, and community practices. Below, we provide actionable recommendations, directly linked to detailed empirical findings in § 4.

Platform Providers. Platform providers should treat security as a foundational requirement for LLM deployments, embedding the "secure by default" principle from the earliest stages of product design. Our analysis in § 4.1.4 and § 4.1.5 highlights widespread insecure defaults, such as the lack of TLS on non-standard ports and extensive certificate reuse, which enable attackers to intercept or spoof communications. Providers must enforce authentication and access control on all public-facing APIs and management interfaces by default, as recommended in § 4.2.1, and absolutely prohibit unauthenticated exposure to the internet. To address risks like model information disclosure

and resource abuse observed in § 4.3.1 and § 4.3.3, providers should offer "one-click hardening" tools and user-friendly security guides, making it easy to disable sensitive endpoints, restrict access, and enable logging by default. Automated masking of sensitive information in API responses should be implemented to prevent inadvertent leakage. Timely distribution of security patches is essential, especially given the rapid pace of vulnerability discovery.

Developers and Self-Hosted Users. Developers and self-hosted users must give high priority to access management and privilege separation when deploying LLM services, strictly avoiding the use of default or insecure configurations in production environments. Our findings in RQ2 (§ 4.2) and RQ3 (§ 4.3) reveal that permissive defaults and inadequate isolation are primary causes of exposure. It is strongly recommended to enforce strict API and management port whitelisting, enable strong authentication, and implement role-based access controls for sensitive operations such as inference and model management. Security vulnerabilities in all components should be continuously monitored and patched (§ 4.1.3), with frameworks and dependencies kept up to date. During development and debugging, internal and external network environments should be isolated; unnecessary ports and debug interfaces should be closed, and exposure to external scans minimized (see clustering patterns in § 4.1.2). When using third-party plugins or models, users should verify the legitimacy of their sources, and prioritize community-audited or platform-certified versions (§ 4.3.4) to reduce the risk of malicious code or backdoors.

5.2 Limitations

Generalizability of Findings. Although the study analyzes 320,102 public-facing LLM services across 15 frameworks, this selection may not fully represent the entire LLM deployment ecosystem. Frameworks such as proprietary or less popular self-hosted solutions may exhibit different security practices or vulnerabilities. Thus, our findings may disproportionately reflect the characteristics of widely adopted frameworks. Additionally, our approach, based on known signatures, may overlook novel or less-documented LLM deployments, potentially leading to an underestimation of the diversity and risks in the broader ecosystem. Future work could explore more adaptive or behavioral measurement strategies to improve coverage of emerging LLM deployments. Furthermore, we are unable to reliably categorize LLM instances according to their application style (e.g., enterprise, consumer, hobbyist) due to limited contextual information in our dataset. As a result, our analysis does not distinguish whether certain security exposures are more prevalent or consequential in specific usage scenarios. Future work could explore more adaptive or behavioral measurement strategies, as well as richer contextual inference, to improve coverage of emerging LLM deployments and enable a finer-grained assessment of security risk by application type.

Potential False Positives and Misclassification. Framework detection relies on FOFA queries and heuristic matching of features such as default ports, HTTP headers, and API metadata. While we took extensive measures to refine these queries and validate results, there remains a possibility of false positives (incorrectly identifying a service as belonging to a framework) or misclassification of frameworks due to overlapping characteristics. In addition, the manual annotation of endpoint risks involved a degree of subjectivity, but disagreements were resolved through discussion and adjudication to help minimize inconsistency.

Potential Temporal Bias The measurement was conducted in April 2025, and deployment practices, security configurations, and framework designs evolve rapidly. For example, frameworks may release security updates or change default configurations, potentially reducing the relevance of some findings. Future studies should evaluate how these trends change over time.

Scope of Security Analysis. This study primarily characterizes the exposure of public-facing LLM endpoints and the types of information they may leak. We did not attempt end-to-end exploitation, adversarial prompt engineering, or other sophisticated attack scenarios that might

more fully demonstrate the practical risks enabled by these exposures. As such, our findings highlight the potential for misuse rather than providing a comprehensive evaluation of real-world exploitability. Future work could build on our results by conducting controlled experiments or red-team assessments to demonstrate concrete attack vectors and their impact.

5.3 Ethical Considerations

This study was conducted with a strong commitment to ethical research practices, ensuring that all activities minimized impact and respected the privacy of service operators. We exclusively probed publicly accessible endpoints using non-destructive, read-only API requests, avoiding any actions that could bypass authentication, access private data, or exploit vulnerabilities. To prevent disruption, all requests were rate-limited and carefully crafted to adhere to documented API standards, ensuring that no harm was caused to the underlying services. Sensitive findings, such as misconfigurations or exposed endpoints, were handled responsibly through coordinated vulnerability disclosure practices, notifying affected framework developers or service operators when appropriate. Additionally, no personally identifiable information (PII) was collected during the study, and any metadata gathered, such as IP addresses or server configurations, was anonymized and aggregated to prevent the identification of specific entities.

6 Related Work

6.1 LLM Safety and Privacy

Recent studies focused on potential safety and privacy risks of LLMs, including adversarial attacks [46], jailbreaking [65], and backdoor attacks [68]. Adversarial attacks targeting LLMs have revealed significant risks to their robustness and alignment. For instance, Zou et al. propose universal and transferable adversarial suffixes capable of bypassing safety mechanisms across multiple LLMs, including black-box models like ChatGPT and Bard [70], while Liu et al. introduce HouYi, a novel black-box prompt injection technique that exploits vulnerabilities in LLM-integrated applications, uncovering risks such as context manipulation and arbitrary payload execution [29]. Building on these findings, jailbreak attacks represent another persistent challenge, as these carefully crafted prompts bypass safety mechanisms. Xu et al. systematically evaluate jailbreak attacks and defenses, showing how various attack techniques exploit LLM vulnerabilities in models like Vicuna and GPT-3.5 [61]. Wei et al. delve deeper into the failure modes of safety training, such as competing objectives and mismatched generalization, which facilitate jailbreaks even in extensively red-teamed models [56]. Additionally, Shen et al. analyze in-the-wild jailbreak prompts using a large-scale dataset, highlighting the persistence and evolution of these attacks over time [47]. Additionally, backdoor attacks have emerged as a critical area of concern, especially in customized or fine-tuned LLMs. Cheng et al. propose TrojanRAG, a backdoor attack framework targeting Retrieval-Augmented Generation (RAG) systems by leveraging contrastive learning and knowledge graphs to create robust triggers [5]. Expanding on this, Zhang et al. explore instruction backdoor attacks in personalized LLMs, demonstrating their effectiveness across multiple datasets and stealth levels [66]. Similarly, Yan et al. introduce CodeBreaker, a framework that embeds undetectable malicious payloads into code completion models, emphasizing the risks posed by fine-tuning on compromised data [62]. These studies primarily focus on the safety and privacy risks inherent to the models themselves.

6.2 Security of LLM Applications

Recent advances have exposed a diverse range of security risks in LLM-integrated applications and services, spanning both application-level attacks and deployment-level vulnerabilities. At the

application layer, researchers have shown that LLM-driven Web Agents are susceptible to indirect prompt injection attacks: malicious instructions embedded in external web content can reliably manipulate agent behaviors, resulting in unintended or unauthorized actions [19, 22, 57]. Pesati et al.[42] further identified critical risks in LLM-integrated applications, including prompt injection, insecure output handling, excessive agency, and supply chain vulnerabilities. Attackers can exploit LLM integrations with web tools, APIs, or retrieval systems to trigger unauthorized actions or leak data[6, 31]. Beyond application logic, the underlying serving architectures introduce new attack surfaces. For example, multi-tenant LLM deployments that share Key-Value (KV) caches, such as vLLM, can inadvertently leak cross-tenant information; attacks like PROMPTPEEK [59] have demonstrated how adversaries can reconstruct other users' prompts via cache analysis. The integration of LLMs with plugins, autonomous agents, and retrieval-augmented generation (RAG) pipelines further expands the attack surface, exposing systems to data poisoning, unauthorized API calls, and model evasion [27, 58, 63]. Recent surveys [20, 64] emphasize that LLM-based agents are particularly prone to security and privacy threats due to their complex and dynamic interactions with external systems. While previous studies have focused on application-layer threats or isolated vulnerabilities, our work is unique in systematically measuring and characterizing the real-world exposure and security posture of public LLM deployments at Internet scale.

Internet-exposed services such as web applications, APIs, and traditional application servers are well known to face risks like injection attacks, authentication bypass, and data leaks [41]. However, LLM deployments differ fundamentally by enabling dynamic, content-driven interactions that introduce new security challenges. The ability of LLMs to interpret and generate complex instructions in natural language gives rise to attack vectors such as prompt injection, while properties like context-sensitive reasoning and integration with diverse external tools can make vulnerabilities easier to exploit and their consequences more far-reaching than in conventional services. Motivated by these distinctive risks, our study aims to provide a comprehensive assessment of the exposure, attack surfaces, and security challenges unique to public LLM deployments.

7 Conclusion

This work presents a comprehensive view of the current landscape of public-facing LLM deployments, offering the first large-scale empirical evidence across 320,102 services spanning 15 frameworks. Our analysis highlights both the rapid expansion and decentralization of self-hosted LLM ecosystems and the widespread presence of security risks, including model disclosure, system configuration leakage, unauthorized access and resource abuse, vulnerabilities, and sensitive content generation. These findings reveal critical gaps between deployment practices and security requirements. Moving forward, we will extend our study with temporal analysis of deployment trends and more detailed assessments of security and operational risks in real-world LLM ecosystems.

Data Availability Statement

To promote transparency and reproducibility, the artifact is publicly accessible at https://github.com/security-pride/LLM_Deployment.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (grants No.62572209, 62502168) and the Hubei Provincial Key Research and Development Program (grant No. 2025BAB057).

References

[1] Alpha Lab of Topsec. 2025. Security Report on Vulnerabilities in Large Language Model Components and Application

Threats.
[2] Anonymous. 2024. Awesome Free Ollama. https://freeollama.oneplus1.top.
[3] Eduardo Berlanga. 2024. RCE via pickle deserialization (unpickling) in comfyanonymous/comfyui. https://huntr.com/bounties/fb5dce86-ca63-40aa-9ff9-c38419a79010.
[4] Zhenpeng Chen, Yanbin Cao, Yuanqiang Liu, Haoyu Wang, Tao Xie, and Xuanzhe Liu. 2020. A comprehensive study on challenges in deploying deep learning based software. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 750–762. doi:10.1145/3368089.3409759
[5] Pengzhou Cheng, Yidong Ding, Tianjie Ju, Zongru Wu, Wei Du, Ping Yi, Zhuosheng Zhang, and Gongshen Liu. 2024. TrojanRAG: Retrieval-Augmented Generation Can Be Backdoor Driver in Large Language Models. *CoRR* abs/2405.13401 (2024). arXiv:2405.13401 doi:10.48550/ARXIV.2405.13401
[6] Jeffrey Yang Fan Chiang, Seungjae Lee, Jia-Bin Huang, Furong Huang, and Yizheng Chen. 2025. Why are web ai agents more vulnerable than standalone llms? a security analysis. *arXiv preprint arXiv:2502.20383* (2025).
[7] Silicon Cloud. 2025. Silicon Flow. https://docs.siliconflow.cn/en/userguide/introduction.
[8] White Hat Community. 2024. Fofa: Cyberspace Asset Search Engine. https://fofa.info/.
[9] NextChat Contributors. 2024. NextChat. https://github.com/ChatGPTNextWeb.
[10] Github Advisory Database. 2024. ComfyUI-Impact-Pack is vulnerable to Path Traversal. https://github.com/advisories/GHSA-6mx8-m8xp-f2vc.
[11] DeepSeek-AI. 2025. DeepSeek LLM. https://github.com/deepseek-ai.
[12] ComfyUI Developers. 2024. ComfyUI. https://github.com/comfyanonymous/ComfyUI.
[13] Volcano Engine. 2025. VolcanoArk. https://www.volcengine.com/product/ark.
[14] Sead Fadilpašić. 2025. Hundreds of LLM servers left exposed online - here's what we know. https://www.techradar.com/pro/security/hundreds-of-llm-servers-left-exposed-online-heres-what-we-know.
[15] OWASP Foundation. 2024. OWASP Top 10 for LLM Applications 2025. https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/.
[16] Wensheng Gan, Shicheng Wan, and Philip S. Yu. 2023. Model-as-a-Service (MaaS): A Survey. In *IEEE International Conference on Big Data, BigData 2023, Sorrento, Italy, December 15-18, 2023*, Jingrui He, Themis Palpanas, Xiaohua Hu, Alfredo Cuzzocrea, Dejing Dou, Dominik Slezak, Wei Wang, Aleksandra Gruca, Jerry Chun-Wei Lin, and Rakesh Agrawal (Eds.). IEEE, 4636–4645. doi:10.1109/BIGDATA59044.2023.10386351
[17] Georgi Gerganov. 2024. llama.cpp. https://github.com/ggerganov/llama.cpp.
[18] Elio Biasiotto Giannis Tziakouris. 2025. Detecting Exposed LLM Servers: A Shodan Case Study on Ollama. https://blogs.cisco.com/security/detecting-exposed-llm-servers-shodan-case-study-on-ollama.
[19] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security* (Copenhagen, Denmark) *(AISec '23)*. Association for Computing Machinery, New York, NY, USA, 79–90. doi:10.1145/3605764.3623985
[20] Feng He, Tianqing Zhu, Dayong Ye, Bo Liu, Wanlei Zhou, and Philip S Yu. 2024. The emerged security and privacy of llm agent: A survey with case studies. *arXiv preprint arXiv:2407.19354* (2024).
[21] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Trans. Softw. Eng. Methodol.* 33, 8, Article 220 (Dec. 2024), 79 pages. doi:10.1145/3695988
[22] Faria Huq, Jeffrey P Bigham, and Nikolas Martelaro. 2023. What's important here?: Opportunities and challenges of LLM in retrieving information from web interface. *R0-FoMo: Robustness of Few-shot and Zero-shot Learning in Large Foundation Models* (2023).
[23] Anyscale Inc. 2024. Ray Serve. https://docs.ray.io/en/latest/serve/index.html.
[24] Timur Isachenko and Shamim Bhuiyan. 2024. *Generative AI with local LLM.* Timur Isachenko.
[25] Project Jupyter. 2024. Jupyter Notebook. https://jupyter.org/.
[26] Malgorzata Lazuka, Andreea Anghel, and Thomas Parnell. 2024. LLM-Pilot: Characterize and Optimize Performance of your LLM Inference Services. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–18.
[27] Ang Li, Yin Zhou, Vethavikashini Chithrra Raghuram, Tom Goldstein, and Micah Goldblum. 2025. Commercial LLM Agents Are Already Vulnerable to Simple Yet Dangerous Attacks. *arXiv preprint arXiv:2502.08586* (2025).
[28] Baolin Li, Yankai Jiang, Vijay Gadepally, and Devesh Tiwari. 2024. Llm inference serving: Survey of recent advances and opportunities. *arXiv preprint arXiv:2407.12391* (2024).
[29] Yi Liu, Gelei Deng, Yuekang Li, Kailong Wang, Tianwei Zhang, Yepang Liu, Haoyu Wang, Yan Zheng, and Yang Liu. 2023. Prompt Injection attack against LLM-integrated Applications. *CoRR* abs/2306.05499 (2023). arXiv:2306.05499 doi:10.48550/ARXIV.2306.05499

[30] Avi Lumelsky. 2024. More Models, More ProbLLMs. https://www.oligo.security/blog/more-models-more-probllms.

[31] Fazel Mohammad Ali Pour and Mohammadreza Rashidi. 2024. Web LLM Attacks: Unveiling the Future of Cyber Threats. *Available at SSRN 5049058* (2024).

[32] NIST National Vulnerability Database. 2024. Apache HTTP Server 2.4.29 Vulnerabilities. https://nvd.nist.gov/vuln/detail/CVE-2019-0211.

[33] NIST National Vulnerability Database. 2024. Nginx 1.14.0 Vulnerabilities. https://nvd.nist.gov/vuln/detail/CVE-2019-20372.

[34] Nomic-AI. 2024. GPT4All. https://github.com/nomic-ai/gpt4all.

[35] NSFOCUS. 2025. Ollama Unauthorized Access Vulnerability Due to Improper Configuration (CNVD-2025-04094). https://nsfocusglobal.com/ollama-unauthorized-access-vulnerability-due-to-misconfiguration-cnvd-2025-04094/.

[36] National Vulnerability Database (NVD). 2024. CVE-2024-37032: Ollama Remote Code Execution Vulnerability. https://nvd.nist.gov/vuln/detail/CVE-2024-37032.

[37] National Vulnerability Database (NVD). 2024. CVE-2024-6707: OpenWebUI Arbitrary File Upload Vulnerability. https://nvd.nist.gov/vuln/detail/CVE-2024-6707.

[38] oobabooga. 2024. Text Generation Web UI. https://github.com/oobabooga/text-generation-webui.

[39] OpenAI. 2025. Build leading AI products on OpenAI's platform. https://openai.com/api/.

[40] OpenAI. 2025. Introducing gpt-oss. https://openai.com/index/introducing-gpt-oss/.

[41] OWASP. 2025. OWASP Top Ten Web Application Security Risks. https://owasp.org/www-project-top-ten/.

[42] Nikhil Pesati. 2024. Security Considerations for Large Language Model Use: Implementation Research in Securing LLM-Integrated Applications. *Available at SSRN 4962370* (2024).

[43] Cosmopolitan Project. 2024. Llamafile. https://github.com/Mozilla-Ocho/llamafile.

[44] Sebastián Ramírez. 2024. FastAPI/Swagger UI. https://fastapi.tiangolo.com/.

[45] Elliot Ward Raul Onitza-Klugman, Rory McNamara. 2025. Don't Get Too Comfortable: Hacking ComfyUI Through Custom Nodes. https://labs.snyk.io/resources/hacking-comfyui-through-custom-nodes/.

[46] Erfan Shayegani, Md Abdullah Al Mamun, Yu Fu, Pedram Zaree, Yue Dong, and Nael B. Abu-Ghazaleh. 2023. Survey of Vulnerabilities in Large Language Models Revealed by Adversarial Attacks. *CoRR* abs/2310.10844 (2023). arXiv:2310.10844 doi:10.48550/ARXIV.2310.10844

[47] Xinyue Shen, Zeyuan Chen, Michael Backes, Yun Shen, and Yang Zhang. 2024. "Do Anything Now": Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 1671–1685. doi:10.1145/3658644.3670388

[48] AnythingLLM Team. 2024. AnythingLLM. https://github.com/Mintplex-Labs/anything-llm.

[49] Gradio Team. 2024. Gradio. https://www.gradio.app/.

[50] Jan Team. 2024. Jan. https://github.com/janhq/jan.

[51] Ollama Team. 2024. Ollama. https://ollama.com.

[52] Open WebUI Team. 2024. Open WebUI. https://github.com/open-webui/open-webui.

[53] UpGuard Team. 2025. Understanding and Securing Exposed Ollama Instances. https://www.upguard.com/blog/understanding-and-securing-exposed-ollama-instances.

[54] vLLM Team. 2024. vLLM. https://github.com/vllm-project/vllm.

[55] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers Comput. Sci.* 18, 6 (2024), 186345. doi:10.1007/S11704-024-40231-1

[56] Alexander Wei, Nika Haghtalab, and Jacob Steinhardt. 2023. Jailbroken: How Does LLM Safety Training Fail?. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/fd6613131889a4b656206c50a8bd7790-Abstract-Conference.html

[57] Fangzhou Wu, Shutong Wu, Yulong Cao, and Chaowei Xiao. 2024. Wipi: A new web threat for llm-driven web agents. *arXiv preprint arXiv:2402.16965* (2024).

[58] Fangzhou Wu, Ning Zhang, Somesh Jha, Patrick McDaniel, and Chaowei Xiao. 2024. A new era in llm security: Exploring security concerns in real-world llm-based systems. *arXiv preprint arXiv:2402.18649* (2024).

[59] Guanlong Wu, Zheng Zhang, Yao Zhang, Weili Wang, Jianyu Niu, Ye Wu, and Yinqian Zhang. 2025. I Know What You Asked: Prompt Leakage via KV-Cache Sharing in Multi-Tenant LLM Serving. In *Proceedings of the 2025 Network and Distributed System Security (NDSS) Symposium. San Diego, CA, USA.*

[60] Qianqian Xie, Dong Li, Mengxi Xiao, Zihao Jiang, Ruoyu Xiang, Xiao Zhang, Zhengyu Chen, Yueru He, Weiguang Han, Yuzhe Yang, et al. 2024. Open-FinLLMs: Open Multimodal Large Language Models for Financial Applications. *arXiv preprint arXiv:2408.11878* (2024).

[61] Zihao Xu, Yi Liu, Gelei Deng, Yuekang Li, and Stjepan Picek. 2024. A Comprehensive Study of Jailbreak Attack versus Defense for Large Language Models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 7432–7449. doi:10.18653/V1/2024.FINDINGS-ACL.443

[62] Shenao Yan, Shen Wang, Yue Duan, Hanbin Hong, Kiho Lee, Doowon Kim, and Yuan Hong. 2024. An LLM-Assisted Easy-to-Trigger Backdoor Attack on Code Completion Models: Injecting Disguised Vulnerabilities against Strong Detection. In *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/yan

[63] Hongwei Yao, Haoran Shi, Yidou Chen, Yixin Jiang, Cong Wang, Zhan Qin, Kui Ren, and Chun Chen. 2025. ControlNET: A Firewall for RAG-based LLM System. *arXiv preprint arXiv:2504.09593* (2025).

[64] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing* (2024), 100211.

[65] Sibo Yi, Yule Liu, Zhen Sun, Tianshuo Cong, Xinlei He, Jiaxing Song, Ke Xu, and Qi Li. 2024. Jailbreak Attacks and Defenses Against Large Language Models: A Survey. *CoRR* abs/2407.04295 (2024). arXiv:2407.04295 doi:10.48550/ARXIV.2407.04295

[66] Rui Zhang, Hongwei Li, Rui Wen, Wenbo Jiang, Yuan Zhang, Michael Backes, Yun Shen, and Yang Zhang. 2024. Instruction Backdoor Attacks Against Customized LLMs. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association, Philadelphia, PA, 1849–1866. https://www.usenix.org/conference/usenixsecurity24/presentation/zhang-rui

[67] Tianyi Zhang, Cuiyun Gao, Lei Ma, Michael Lyu, and Miryung Kim. 2019. An empirical study of common challenges in developing deep learning applications. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 104–115.

[68] Shuai Zhao, Meihuizi Jia, Zhongliang Guo, Leilei Gan, Xiaoyu Xu, Xiaobao Wu, Jie Fu, Yichao Feng, Fengjun Pan, and Luu Anh Tuan. 2025. A Survey of Recent Backdoor Attacks and Defenses in Large Language Models. arXiv:2406.06852 [cs.CR] https://arxiv.org/abs/2406.06852

[69] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2025. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL] https://arxiv.org/abs/2303.18223

[70] Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. 2023. Universal and Transferable Adversarial Attacks on Aligned Language Models. *CoRR* abs/2307.15043 (2023). arXiv:2307.15043 doi:10.48550/ARXIV.2307.15043

Received October 2025; revised December 2025; accepted January 2026