Unveiling the Landscape of LLM Deployment in the Wild: An Empirical Study

Xinyi Hou*, Jiahao Han*, Yanjie Zhao, and Haoyu Wang†

Huazhong University of Science and Technology, Wuhan, China

xinyihou@hust.edu.cn, jiahaohan789@gmail.com, yanjie_zhao@hust.edu.cn, haoyuwang@hust.edu.cn

Abstract—*Background*: Large language models (LLMs) are increasingly deployed via open-source and commercial frameworks, enabling individuals and organizations to self-host advanced AI capabilities. However, insecure defaults and misconfigurations often expose LLM services to the public Internet, posing significant security and system engineering risks. *Aims*: This study aims to unveil the current landscape of public-facing LLM deployments in the wild through a large-scale empirical study, focusing on service prevalence, exposure characteristics, systemic vulnerabilities, and associated risks. *Method*: We conducted an Internet-wide measurement to identify public-facing LLM deployments across 15 frameworks, discovering 320,102 services. We extracted 158 unique API endpoints, grouped into 12 functional categories based on capabilities and security risks. We further analyzed configurations, authentication practices, and geographic distributions, revealing deployment trends and systemic issues in real-world LLM system engineering. *Results*: Our study shows that public LLM deployments are rapidly growing but often insecure. Among all endpoints, we observe widespread use of insecure protocols, poor TLS configurations, and unauthenticated access to critical operations. Security risks, including model disclosure, system leakage, and unauthorized access, are pervasive, highlighting the need for secure-by-default frameworks and stronger deployment practices. *Conclusions*: Public-facing LLM deployments suffer from widespread security and configuration flaws, exposing services to misuse, model theft, resource hijacking, and remote exploitation. Strengthening default security, deployment practices, and operational standards is critical for the growing self-hosted LLM ecosystem.

Index Terms—large language models, LLM, LLM deployment, empirical study

I. INTRODUCTION

The rapid adoption of large language models (LLMs), driven by prominent models such as the GPT series from OpenAI [33] and DeepSeek's open-source variants [1], has profoundly reshaped the landscape of artificial intelligence (AI) applications. **Once confined primarily to research labs and industrial environments, these models have increasingly become accessible to the general public, fueling a widespread trend towards self-hosted and open-source deployments** [18]. The availability of user-friendly tools and vibrant community ecosystems [42], [43], [38] has empowered individual enthusiasts, small enterprises, and developers to independently deploy and customize powerful language models for various personal and professional purposes, such as creative writing and content creation [21], software development

and maintance [17], financial analysis and automated investment assistance [48] and personal productivity tools, greatly enriching their daily digital experiences. However, as barriers to LLM deployment continue to fall, more deployments occur without rigorous security considerations, exposing users and organizations to new operational and security risks.

Among these concerns, the OWASP Top 10 for LLM Applications 2025 [13] identifies several risks that deserve particular attention during deployment, including sensitive information disclosure, unbounded consumption, and supply chain risks. These risks are especially relevant in self-hosted and open-source scenarios, where models, APIs, and supporting infrastructure are often exposed to the public internet without sufficient protection. These deployment challenges highlight not only the security concerns of LLM applications but also broader software engineering issues around system configuration, exposure management, and operational robustness.

Open-source frameworks commonly used for self-hosted LLM deployments often suffer from insecure default settings and misconfigurations, **exposing sensitive interfaces to the public Internet** without adequate protection and significantly enlarging the attack surface. Such services can be easily discovered via common asset-discovery tools like FOFA [9], Shodan [26], and ZoomEye [41]. For instance, Ollama [42], a framework widely utilized for deploying local LLM services, exposes RESTful APIs publicly by default without authentication, enabling unauthorized operations such as model deletion, theft, GPU resource hijacking, and critical remote code execution (e.g., CVE-2024-37032 [30]). Furthermore, publicly maintained platforms aggregating openly accessible Ollama services [4] exacerbate the issue. Similarly, OpenWebUI [43], commonly integrated alongside Ollama for enhanced interaction capabilities, has suffered from vulnerabilities allowing arbitrary file uploads (CVE-2024-6707 [31]), potentially facilitating remote command execution. Additionally, ComfyUI [11], known for its plugin-rich environment supporting diffusion-based generation tasks, has experienced multiple severe plugin-related security issues, including unauthorized remote code execution, arbitrary file access, and serialization vulnerabilities. Together, these vulnerabilities highlight the urgent need for stronger security awareness and practices in open-source LLM deployments.

Motivated by these observations, we conduct a large-scale empirical analysis to systematically assess the prevalence and

---

*Xinyi Hou and Jiahao Han contributed equally to this work.

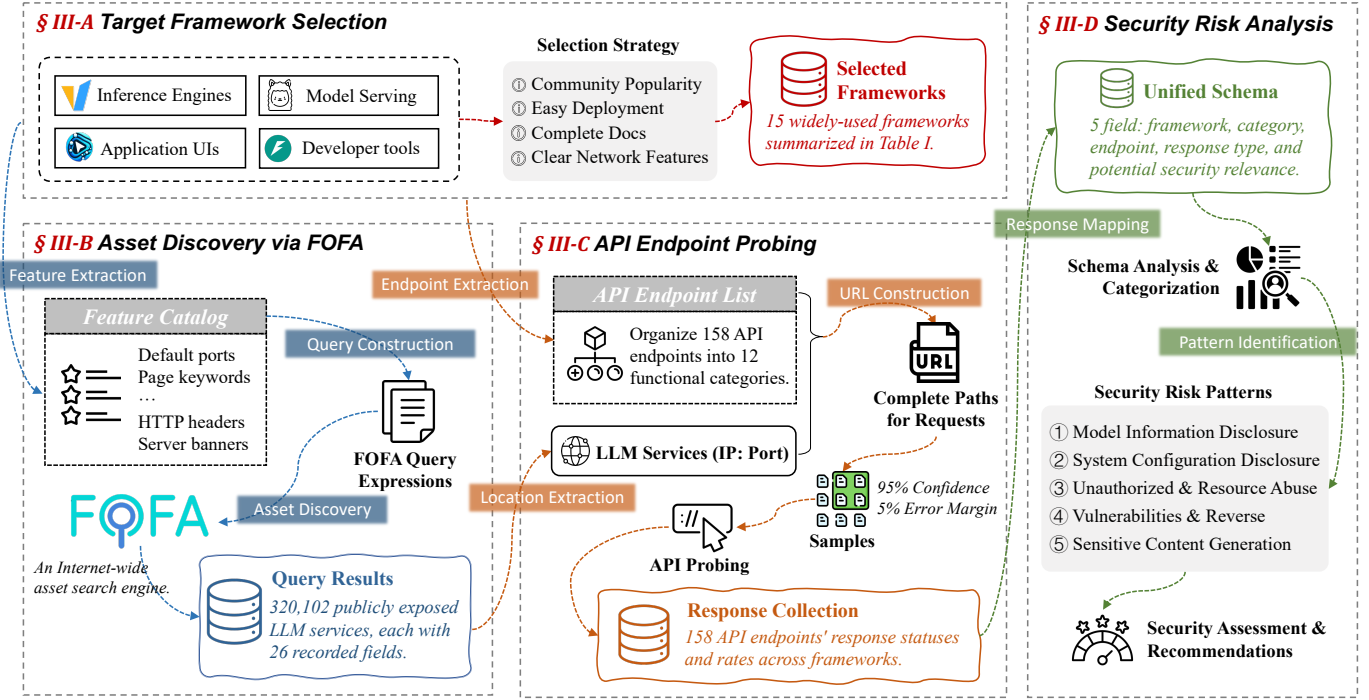†Haoyu Wang is the corresponding author (haoyuwang@hust.edu.cn).

Fig. 1: Overview of the Empirical Analysis Pipeline.

characteristics of public-facing LLM deployments. Our overall analysis pipeline is illustrated in Figure 1. We identify and analyze 320,102 LLM services across the Internet, spanning 15 popular deployment frameworks. From these, we extract 158 unique API endpoints, which we group into 12 functional categories based on their capabilities and associated security risks. Our study analyzes the geographic and network distribution of LLM deployments, uncovers widespread insecure configurations such as unauthenticated inference and model enumeration, and provides practical recommendations for developers and operators. The findings expose widespread security and deployment flaws, highlighting the need for secure-by-default designs and improved operational practices. Our artifacts related to this study are publicly available at https://anonymous.4open.science/r/Public-LLM-Services.

Our main contributions are as follows:

- **Large-Scale Empirical Study:** We identify 320,102 publicly accessible LLM services, providing the first empirical evidence of their global presence, deployment patterns, and exposure characteristics.
- **Exposure and Security Risk Analysis:** We systematically analyze 158 unique API endpoints, categorize them into 12 functional groups, and uncover systemic vulnerabilities in open-source LLM deployments, including model disclosure, system leakage, unauthorized access, vulnerabilities, and sensitive content generation.
- **Practical Recommendations:** We provide actionable guidance for secure-by-default design and deployment, grounded in empirical findings and targeting developers, framework maintainers, and operators.

II. BACKGROUND AND RELATED WORK

A. LLM Deployment Paradigms and Tooling

Deploying LLMs in real-world environments involves a multi-layered stack encompassing computation, scalable serving, user interaction, and developer support. This stack broadly consists of four key components. **Inference Engines.** Inference engines execute LLM computations across diverse hardware. Optimized systems like vLLM [44] and Hugging Face Transformers leverage techniques such as continuous batching, KV cache optimization, and operator fusion to improve throughput and latency [25]. Tools like LLM-Pilot [23] enable benchmarking and configuration across different environments. **Model Serving Frameworks.** Serving frameworks manage hosting, request routing, and scaling. Systems like Ray Serve [20] and Ollama [42] support flexible deployment across cloud and edge platforms [7], [5]. Split-based deployment further distributes model components to balance performance and privacy [6]. **Application User Interfaces.** User-facing applications integrate LLMs via conversational interfaces, RAG pipelines, and autonomous agents. Frameworks like LangChain and ChatGPT plugins simplify this integration but also introduce risks such as indirect prompt injection [45], [19]. **Developer Tools and Ecosystems.** Developer tools support fine-tuning, evaluation, and maintenance. Libraries such as DeepSpeed, Hugging Face PEFT, and LoRA enable efficient model adaptation [18], [37], while tools like TestGen-LLM [3] automate engineering workflows. Specialized accelerators are also emerging to meet large-scale inference demands [12], [7]. Despite growing modularity and accessibility, little is known about how LLM deployments are configured and exposed in

TABLE I: Publicly Exposed LLM Deployment Frameworks Discovered via FOFA (As of April 20, 2025).

Category	Framework	Description	Key Feature	Count
Inference Engines	vLLM [44]	High-throughput, memory-optimized inference service	title/vLLM; port=8000	6,077
	llama.cpp [14]	C/C++ quantized inference engine	title/llama.cpp; port=8080	4,234
	GPT4All [2]	Local GPT model runtime exposing REST endpoint	title/GPT4All; port=8080	2,572
	Llamafile [35]	Single-file GPT execution tool	title/Llamafile; port=8080	39
Model Serving	Ollama [42]	Cross-platform CLI for local LLM API service	header/Ollama; port=11434	155,423
	AnythingLLM [38]	Local knowledge base integration with LLM API	title/AnythingLLM; port=3000	3,766
	Ray Serve [20]	Scalable microservice framework with autoscaling	title/Ray Serve; port=8000	365
Application UIs	Open WebUI [43]	Ollama/GPT-API Web dashboard	title/Open WebUI; port=8080	37,242
	Jan [40]	Interactive local chat UI	title/Jan; port=3000	28,445
	NextChat [10]	Local ChatGPT-style interface	title/chatgpt-next-web; port=3000	25,883
	ComfyUI [11]	Visual workflow builder for LLM and image pipelines	title/ComfyUI; port=8188	15,219
	Gradio [39]	Python toolkit for shareable LLM web demos	title/Gradio; port=7860	9,729
	Text Generation Web UI [32]	Generic LLM Web interface	title/text-generation-webui; port=7860	2,051
Developer Tools	Jupyter Notebook [22]	Interactive Python notebook environment	body/Jupyter Notebook	24,531
	FastAPI/Swagger UI [36]	Auto-generated API docs and interactive endpoints	title/FastAPI; port=8000	4,526
Total				**320,102**

practice. To bridge this gap, we conduct the first large-scale analysis of publicly accessible LLM services in the wild.

B. Security Challenges in LLM Deployment

Large-scale LLM deployments introduce security challenges across networking, authentication, input handling, and resource management layers. A common risk stems from accidental exposure, where open ports or permissive configurations allow unrestricted access to LLM services. Measurements of Internet-facing deployments [45] reveal widespread misconfigurations, leading to unauthorized interactions, resource abuse, and data leakage. Authentication weaknesses further exacerbate these risks. Pesati et al. [34] found that many public LLM services lack proper authentication or rely on fragile mechanisms, exposing them to prompt manipulation, session hijacking, and privilege escalation. Input manipulation, particularly prompt injection, presents another critical threat. Indirect injections, where malicious inputs are embedded in external content, can subvert model behavior without user awareness [45], [15], [19]. Attackers can exploit LLM integrations with web tools, APIs, or retrieval systems to trigger unauthorized actions or leak data [27], [8]. Multi-tenant serving architectures introduce side-channel vulnerabilities. Sharing Key-Value (KV) caches among users, as in vLLM, can leak cross-tenant information. Attacks like PROMPT-PEEK [47] demonstrate that adversaries can reconstruct other users' prompts by analyzing cache access patterns. Furthermore, integrating LLMs with plugins, autonomous agents, and retrieval-augmented generation (RAG) pipelines significantly expands the attack surface [46], [49], [24]. Vulnerabilities in these systems can lead to data poisoning, unauthorized API calls, and model evasion. Surveys [50], [16] highlight that LLM-based agents are especially prone to security and privacy threats due to their complex interactions with external systems. Motivated by these risks, we systematically investigate real-world LLM deployments to uncover prevalent insecure practices and highlight critical gaps in existing security postures.

III. METHODOLOGY

Figure 1 outlines our four-step methodology for analyzing LLM deployments in the wild. We first select representative deployment frameworks (§ III-A), then discover publicly accessible instances via FOFA (§ III-B), probe their APIs to collect metadata (§ III-C), and finally analyze their configurations and security posture (§ III-D).

A. Target Framework Selection

To structure our measurement and ensure broad coverage across the LLM deployment stack, we selected representative frameworks spanning four functional categories, based on the typical architecture of real-world LLM deployments: inference engines, model serving frameworks, application user interfaces (UIs), and developer tools. **Inference engines** (e.g., vLLM [44], llama.cpp [14]) handle local model execution optimized for performance. **Model serving frameworks** (e.g., Ollama [42], Ray Serve [20]) expose scalable APIs. **Application UIs** (e.g., Open WebUI [43], Jan [40], ComfyUI [11]) provide user-facing interfaces, while **developer tools** (e.g., Jupyter Notebook [22], FastAPI [36]) facilitate development workflows and integration. Frameworks were selected based on popularity in open-source communities, ease of deployment, documentation availability, and the distinctiveness of their network exposure characteristics. In total, we studied 15 widely-used frameworks, summarized in Table I.

B. Asset Discovery via FOFA

We use FOFA, a widely adopted Internet-wide asset search engine that indexes IP addresses, domains, and service metadata [9], to discover public-facing LLM deployments at scale. We constructed a feature catalog for each tool, capturing its unique network-level characteristics, to detect these deployments via FOFA. These features include default service ports (e.g., port 11434 for Ollama), page titles or HTML keywords (e.g., `title=''Open WebUI''`), HTTP response headers (e.g., `''Ollama is running''`), and, where available,

favicon hashes or known API paths, as shown in Table I. Based on this catalog, we designed custom FOFA query expressions using the platform's advanced syntax. For instance, services like Ollama were identified using queries such as `app=''Ollama'' && port=''11434''`, while FastAPI-based deployments were located using expressions like `title=''FastAPI'' || body=''FastAPI'') && (port=''8000'' || port=''8080''`.

To improve coverage and reduce false positives, we refined these queries iteratively. This process involved testing against known deployment samples, consulting official documentation, and manually validating a representative subset of results. In particular, we examined rendered landing pages, HTTP headers, and available API metadata to confirm the identity of suspected services. As of April 20, 2025, we had identified 320,102 publicly accessible LLM-related services across 15 representative deployment tools, as shown in Table I. These include 155,423 instances of Ollama, 37,242 of Open WebUI, 28,445 of Jan, and 6,077 of vLLM, among others. These results form the foundation for the subsequent phases of our study, including endpoint probing and configuration analysis, as described in § III-C and § III-D.

TABLE II: API Categories in Exposed LLM Services.

Category	Function	#
Text & Chat Gen	Text and chat completions (OpenAI-style)	23
Embedding Gen	Generate vector embeddings from text	9
Image & Audio	Image generation, editing, speech-to-text, TTS	32
Model Ops	Load, list, delete, and inspect models	28
File Ops	Upload, download, or delete general files	19
Knowledge/RAG	Upload and query knowledge bases for RAG	3
Fine-tuning Tasks	Create fine-tuning jobs and upload training data	10
Session & Kernel	Manage Jupyter sessions, kernels, and clusters	16
Task Queue	Submit tasks, monitor queues, and job scheduling	3
System Config	Get system status, API version, and configuration	6
Moderation Check	Content safety and moderation checks	5
App Deployment	Deploy or manage LLM apps (e.g., Ray Serve)	4
Total		**158**

C. 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 II, 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, including file uploads, session control, and even server-side code execution in certain configurations.** Using these extracted paths, we constructed full URLs by combining them with the network locations of previously discovered LLM services. Probing was conducted via HTTP(S) requests, and all responses

were collected and stored for further analysis. Rather than attempting a comprehensive crawl of all known services, the probing process was designed to focus on identifying insecure deployment practices across representative samples. Sampling was performed within each LLM service category to ensure statistical validity. For each category, we selected services based on a 95% confidence level and a 5% margin of error. This approach allowed us to reduce probing overhead while maintaining representative coverage across deployment types. The resulting dataset provides a comprehensive view of exposed API functionalities and their response behaviors in real-world deployments.

D. Configuration and Security Analysis

Following endpoint probing, we analyzed the returned API responses to assess the configuration and security posture of exposed LLM services. Each response was mapped to a unified schema capturing five key fields: *deployment framework*, *endpoint category*, *endpoint path*, *response type* (e.g., success, denial, error), and *potential security relevance*. This schema enabled consistent comparison across frameworks with heterogeneous designs and naming conventions. The analysis focused on endpoint responsiveness under unauthenticated access, functional exposure across 12 categories, and the presence of risky operations such as model management, file access, and code execution. By aggregating normalized responses across representative samples, we identified common patterns of misconfiguration, authentication gaps, and inconsistent access control enforcement.

IV. RESULTS

This section presents the results of our Internet-wide empirical analysis of public LLM services. We report the overall exposure, deployment characteristics, configuration patterns, and security risks observed in the collected dataset.

A. General Statistics

We conduct a series of foundational statistical analyses to provide a comprehensive overview of the public landscape of self-hosted LLM services. These analyses aim to uncover the global deployment scale, organizational involvement, domain exposure, deployment surfaces, and security postures, forming a basis for deeper security investigations.

1) Global and Organizational Deployment Trends: We analyze 320,012 exposed LLM service endpoints, aggregating their origin by country and associated hosting organization. Figure 2 illustrates this landscape with two pie charts summarizing the top contributors by country and organization.

Geographic Centralization. As shown in Figure 2a, the United States is by far the dominant origin of public LLM services, with 111,728 instances, more than twice the count of the second-largest country, 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.
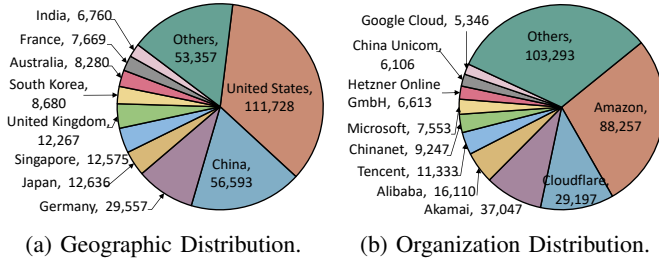
(a) Geographic Distribution.

(b) Organization Distribution.

Fig. 2: Global Landscape of Public LLM Deployment.

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. This distribution hints at a growing democratization of LLM capabilities, though one is still heavily shaped by infrastructure access and national regulatory frameworks.

Organizational Dominance and Long Tail. As shown in Figure 2b, a few major providers dominate public LLM deployments. Amazon alone hosts 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 hosted by entities categorized as "Others". This indicates that a substantial number of deployments originate from smaller cloud vendors, university networks, hobbyist servers, or edge nodes. While this decentralization reflects the democratization of LLM deployment, it also introduces significant heterogeneity in operational practices.

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 focusing on a few high-density targets, whereas defenders must cope with a much broader and less predictable surface.

2) High-Traffic Domains and Service Concentration: More than 210,000 exposed LLM services lack valid domain assignments, representing a significant portion of the overall deployment landscape. This reflects widespread deployment without proper DNS configuration or certificate binding, likely resulting from incomplete setup processes or reliance on default settings in automated toolchains. These services are typically accessible only via IP address and represent a considerable portion of deployments with poor post-deployment hygiene, weakening traceability and trust mechanisms.

Beyond these unassigned cases, certain domains are associated with an unusually high number of LLM instances. As shown in Figure 3, domains such as `nellasushi.es`, `mysuccess.be`, and `human-rights-law.eu` each host

thousands of 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[1] 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. 3: High-Traffic Domains by Number of LLM Services.

3) Server Stack Composition: Public LLM services are predominantly hosted using familiar and widely adopted server stacks. As shown in Table III, 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

---

[1]Supplemental heatmap available at https://anonymous.4open.science/r/Public-LLM-Services.

Fig. 4: Distribution of LLM Services Across Ports by Deployment Framework.

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 envi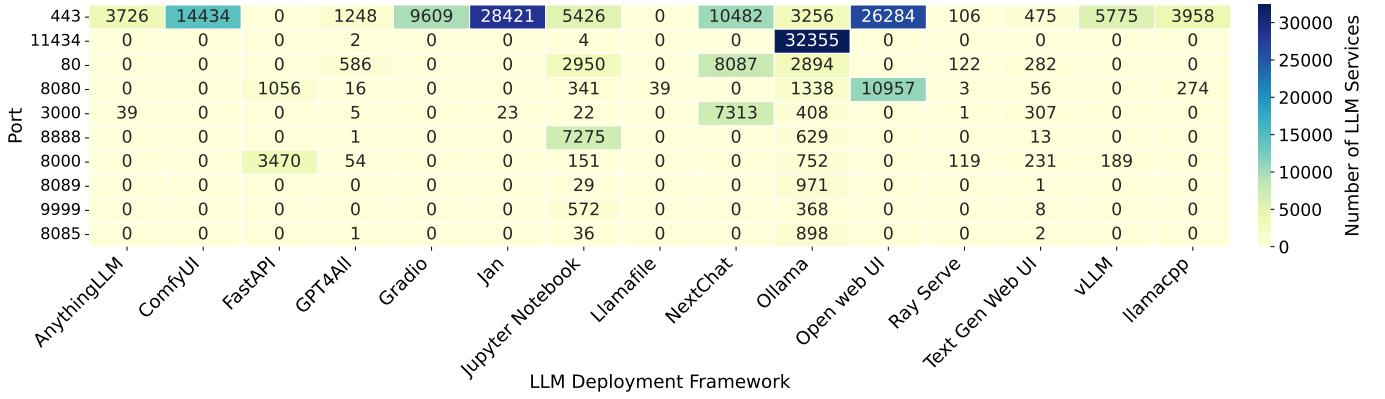ronment metadata. Over 50,000 instances do not report a valid operating system, and more than 90,000 specify only generic server identifiers. These incomplete configurations likely arise from containerized or scripted deployments where base image information is obscured. While such approaches streamline 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 [29], [28]. Lightweight servers such as `Uvicorn` and `TornadoServer`, though efficient, often lack default hardening and are more prone to insecure defaults. The prevalence of opaque and minimalist setups further complicates automated auditing and increases uncertainty in assessing the broader exposure surface.

4) Communication Security and Port Exposure: The communication security of public LLM services varies significantly across ports and deployment frameworks. As shown in Figure 5, 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. These ports are commonly associated with frameworks including Open Web UI, Jan, and Ollama, as shown in Figure 4. 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 cryptographic vulnerabilities.

TABLE III: Top OS–Server Deployment Combinations in Exposed LLM Services.

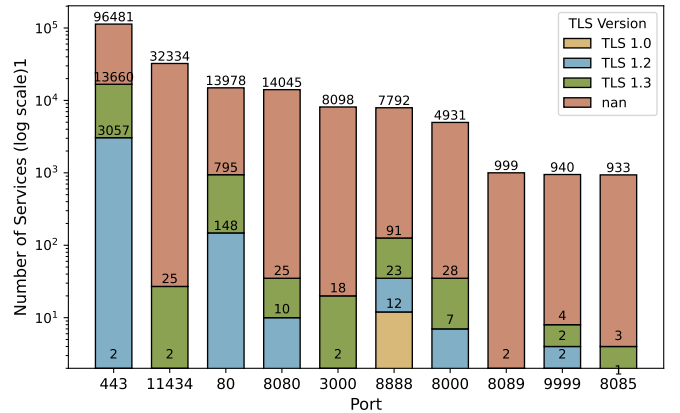| OS	Server	Count	Percentage	Share in OS
ubuntu	nginx/1.18.0 (ubuntu)	7,242	42.84%	53.23%
ubuntu	nginx/1.24.0 (ubuntu)	3,659	21.65%	26.9%
windows	microsoft-iis/10.0	567	3.35%	50.04%
ubuntu	apache/2.4.52 (ubuntu)	545	3.22%	4.01%
debian	apache/2.4.62 (debian)	457	2.70%	44.63%
ubuntu	apache/2.4.41 (ubuntu)	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%

Fig. 5: TLS Version Distribution Across Ports (Log Scale).

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. Without it, even well-intentioned self-hosted setups remain vulnerable by design.

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 6, common names (CNs) such as `localhost` and generic domain labels (e.g., `nellasushi.es`, `mysuccess.be`) appear frequently, while a significant number of entries are simply marked as `nan`, indicating missing or unparsable certificate subject fields. Quantitative analysis 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. While some reuse is expected in containerized or replicated environments, the scale observed here suggests a lack of certificate management hygiene. This undermines the integrity of TLS-based identity validation, as the same certificate may appear in unrelated deployments or across unaffiliated IP ranges.

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. Together, these patterns expose a weak identity layer in the self-hosted LLM ecosystem, undermining encryption's role in both confidentiality and endpoint authentication.

> **Takeaway 1:** *Public LLM deployments are expanding rapidly but exhibit systemic security weaknesses. Among 320,012 endpoints, over 40% use plain HTTP, more than 210,000 have missing or generic TLS metadata, and a few providers (e.g., Amazon, Cloudflare) dominate global exposure. Long-tail deployments across smaller networks add inconsistency and unpredictability. These findings highlight the urgent need for secure-by-default frameworks, better certificate management, and standardized operational practices for the self-hosted LLM ecosystem.*

B. API Responsiveness Analysis

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

1) Framework-Level API Responsiveness: Across exposed LLM deployments, we observe substantial variation in API responsiveness. As shown in Table IV, frameworks like Ollama and Llamafile respond to over 80% of unauthenticated

Fig. 6: Distribution of Reused TLS Certificate Subject CNs.

API requests, suggesting permissive default configurations and minimal access control. In contrast, widely used platforms such as Open WebUI, Jan, and Text Generation WebUI exhibit responsiveness rates below 2%, indicating either frontend-only exposure or stricter backend protection. FastAPI and AnythingLLM are excluded due to the lack of standardized APIs. FastAPI serves as a generic backend framework with fully customizable endpoints, while AnythingLLM exposes inconsistent interfaces across deployments, often relying on UI interactions. Both lack the structural consistency required for comparable measurement.

TABLE IV: Responsiveness of Different LLM Deployment Frameworks.

Framework	# Resp.	Sample	Population	Resp. Rate
Ollama	309	384	155,423	80.47%
Open WebUI	2	381	37,242	0.52%
Jan	5	380	28,445	1.32%
NextChat	1	379	25,883	0.26%
Jupyter Notebook	53	379	24,531	13.98%
ComfyUI	42	375	15,219	11.20%
Gradio	60	370	9,729	16.22%
vLLM	10	362	6,077	2.76%
llama.cpp	36	353	4,234	10.20%
GPT4All	13	335	2,572	3.88%
Text Generation Web UI	2	324	2,051	0.62%
Ray Serve	1	188	365	0.53%
Llamafile	29	36	39	80.56%

The HTTP status codes returned by these services reflect their underlying security posture. While some services return 200, exposing full functionality, many respond with 401 (unauthorized), 403 (forbidden), or 404 (not found), indicating authentication checks, access restrictions, or endpoint obfuscation. A smaller number return 5xx errors, suggesting unstable or misconfigured services. We also observe a notable number of 400 Bad Request responses, mainly from Gradio-based deployments. These indicate that the service is reachable but the API is undocumented, unsupported, or not intended for direct programmatic use, consistent with Gradio's focus on interactive UIs rather than formal APIs. These response patterns reveal implicit security mechanisms across frameworks.

Framework	Text/Chat Gen	Embed Gen	Img/Audio	Model Ops	File Ops	RAG	Fine-tune	Sess./Kernel	Task Queue	Sys Config	Moderation	App Deploy
Ollama	2 / 2[1]	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	

[1] Each cell shows "successful / total" API endpoints in that category (e.g., *2 / 2* means all responded successfully).

High responsiveness is often associated with frameworks that expose APIs by default and lack built-in protection, whereas lower responsiveness generally corresponds to more defensive configurations. Notably, a non-trivial portion of services respond with structured denial (401/403), suggesting that some frameworks implement basic safeguards even when deployed in public environments.

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 V, 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. 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. The observed exposure patterns reveal trade-offs between usability and security. Frameworks intended for local or development use often leak internal APIs by default without authentication or isolation, while production-oriented systems show tighter surface control. Even endpoints related to queue status or system configuration, while not critical, can still cause unintended exposure of internal system states.

3) Per-Endpoint Result Representation: All API responses are first normalized into a unified schema capturing *deployment framework*, *endpoint category*, *endpoint path*, *response type*, and *potential security relevance*. This abstraction enables consistent analysis across frameworks with differing designs

TABLE VI: Responsiveness of Ollama API Endpoints.

Category	Endpoint	Function	# Resp.	Resp. Rate[1]
Text/Chat Gen	/api/generate	Text Completion	220	57.29%
	/api/chat	Chat Completion	220	57.29%
Embedding	/api/embeddings	Generate Embedding	10	2.60%
Model Ops	/api/tags	List Local Models	289	75.26%
	/api/pull	Pull Model	256	66.67%
	/api/push	Push Model	208	54.17%
	/api/delete	Delete Model	109	28.39%
	/api/copy	Copy Model	80	20.83%
	/api/show	Show Model Info	36	9.38%
	/api/create	Create Model	33	8.59%
	/api/running	List Loaded Models	9	2.34%
System Config	/api/version	Get Ollama Version	9	2.34%

Note: Based on 384 sampled LLM service invocations.

and naming conventions. Based on this schema, we analyze Ollama as a representative case. Table VI summarizes endpoint responsiveness across 384 observed invocations. **Text and chat completion** endpoints responded in 57.29% of cases. While often active, their partial availability suggests exposure conditioned on runtime state or configuration, potentially leaking system behavior to unauthenticated clients. The **embedding** endpoint was accessible in only 2.60% of attempts. Though rarely enabled, its occasional exposure still poses risk, as embedding vectors may support inference attacks even in limited contexts. **Model management** endpoints showed uneven availability. Listing local models was frequently possible (75.26%), likely due to its role in coordination. Pull and push operations were moderately available (66.67%, 54.17%), while destructive actions like delete, copy, and create responded in fewer than 30% of cases, suggesting partial lockdowns, though enforcement remains inconsistent. **System-level** endpoints, including version and runtime model queries, responded in under 3% of cases. This likely reflects hardening efforts, but their occasional exposure highlights the need for stricter endpoint visibility controls. The results highlight that in Ollama deployments, many endpoints, including those that expose model metadata, are accessible without authentication, allowing unauthorized users to retrieve sensitive information and revealing a critical lack of access control.

> **Takeaway 2:** *Our findings reveal widespread security weaknesses in real-world LLM deployments. A large portion of services, such as Ollama and Llamafile, allow over 80% of unauthenticated API requests, exposing critical operations like text generation and model management (e.g., 75.26% model listing success). Even when frameworks appear restrictive, inconsistencies and partial exposures persist. This indicates that current deployment practices lack systematic access control and minimal exposure principles, leaving many services vulnerable to unauthorized access, information leakage, and misuse.*

C. Security and Risk Analysis

As shown in Table VII, this section systematically analyzes potential security risks in LLM deployment frameworks. Each subcategory combines observed cases with deeper security insights to reveal not only surface-level exposures but also underlying attack vectors and systemic vulnerabilities.

1) Model Information Disclosure: Model information disclosure is among the most prevalent security risks across LLM deployment frameworks, as indicated in Table VII. For example, the `/show` endpoint can reveal detailed model metadata, while the `/history` endpoint may leak prior inference workflows, outputs, and fine-tuning traces, compromising user privacy and exposing proprietary data. Notably, in Ollama, the `/api/tags` endpoint exhibits an exceptionally high exposure rate of 70.57%, highlighting the severity of this risk even in widely adopted deployment platforms. The `/embeddings` endpoint (1.87% exposure) in ComfyUI further illustrates this problem by exposing the list of loaded textual inversion embeddings, such as *Bad-Hands-XL*, inadvertently revealing deployment purposes. Similarly, in `llama.cpp`, the `/v1/models` 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, prompt injection, and unauthorized access. In severe cases, attackers could reconstruct user intents, manipulate outputs, or compromise the deployment's integrity and confidentiality.

2) Hardware and System Configuration Disclosure: Disclosure of system configuration details further enlarges the attack surface by providing adversaries with valuable intelligence about the underlying environment. ComfyUI's `/system_stats` endpoint (4.53%) reveals operating system types, total and available memory, GPU specifications (e.g., RTX 4090 with 24GB VRAM), and ComfyUI version information. Such granular 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.

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` (1.07%) and `/prompt` (3.20%) allow unauthenticated users to submit inference tasks or monitor system load. The ability to observe task queues enables attackers to perform real-time load sensing, identifying idle periods when resource-draining or prompt injection attacks 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. In the absence of fine-grained access control and rate-limiting, such systems remain highly vulnerable to long-term degradation and operational hijacking.

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` (9.33%) and `/object_info` (1.60%) 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 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. Passive metadata collection can quickly escalate into full system compromise, underscoring the critical risks of seemingly minor exposures.

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 Ollama, LLMs often labeled as "uncensored", can be manipulated through crafted prompts to bypass moderation controls. Similarly, in Text Generation WebUI, insufficient prompt sanitization may allow adversarial inputs to elicit harmful responses or extract proprietary system behaviors. If prompt histories, session contexts, or interaction logs are improperly secured, attackers can retrieve past prompts and outputs to refine injection

TABLE VII: Risk Categorization (Min $\sim$ Max Percentage) of Different LLM Deployment Frameworks.

Framework	Model Info Disclosure	System Config Disclosure	Unauthorized & Abuse	Vulnerabilities & Reverse	Sensitive Content Gen
ComfyUI	$1.87\% \sim 6.67\%$	4.53%	$1.07\% \sim 3.20\%$	$1.60\% \sim 9.33\%$	3.20%
Ollama	$2.08\% \sim 70.57\%$	–	$4.95\% \sim 38.02\%$	–	$45.31\% \sim 45.57\%$
Text Gen WebUI	0.31%	0.31%	–	–	0.31%
GPT4All	2.40%	–	–	–	–
Llamacpp	4.53%	2.27%	–	–	–
Llamafile	16.57%	–	–	–	–
Jupyter Notebook	–	–	–	13.19%	–

[1] Each cell shows the observed minimum $\sim$ maximum percentage of API endpoints posing the corresponding risk category. If minimum equals maximum, only a single value is shown. "–" indicates no relevant risk observed. Detailed case rates for ComfyUI endpoints are summarized in Table VIII.

TABLE VIII: Security Risk Exposure of ComfyUI Endpoints.

Security Risk	API Endpoint	# Cases	Case Rate[1]
Model Information Disclosure	`/embeddings`	7	1.87%
	`/history`	25	6.67%
System Configuration Disclosure	`/system_stats`	17	4.53%
Unauthorized & Resource Abuse	`/queue`	4	1.07%
	`/prompt`	12	3.20%
Vulnerabilities & Reverse	`/object_info`	6	1.60%
	`/extensions`	35	9.33%
Sensitive Content Generation	`/prompt`	12	3.20%

[1] Based on analysis of 375 sampled ComfyUI endpoints. # Cases and Case Rates indicate confirmed security risks..

strategies and escalate misuse. Such manipulations may cause reputational harm, legal exposure, and regulatory penalties. As scrutiny of AI-generated content grows, securing the content generation pipeline is essential to mitigating systemic risks.

> **Takeaway 3:** *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. Notably, 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.*

V. DISCUSSION

A. Root Causes of Interface Exposure

The widespread exposure of LLM deployment interfaces stems from a combination of insecure defaults, weak boundary enforcement, and operational misunderstandings. Many frameworks are designed with the assumption of a trusted local environment, where binding to "localhost" is seen as sufficient. However, in practice, deployments often occur in containerized or multi-user settings, where services are inadvertently exposed to broader networks without authentication or access control. This risk is amplified by a usability-first design culture. To simplify prototyping, many tools expose powerful endpoints by default and provide minimal guidance on secur-

ing them. Documentation often emphasizes quick interaction through HTTP APIs but overlooks the security implications of commands related to model loading, prompt submission, or file manipulation. When authentication options exist, they are frequently disabled by default or poorly documented. Another key factor is the insufficient separation between user-level interactions and system-level control. Endpoints that allow prompt execution, plugin loading, or system inspection are often directly accessible without input validation or role-based restrictions. In extensible platforms, dynamically loaded modules operate without sandboxing, creating opportunities for privilege escalation or internal reconnaissance. Finally, interface exposure is exacerbated by limited user awareness. Developers and deployers often assume that local services are inherently secure, ignoring risks introduced by container networking, port forwarding, or LAN visibility. Without monitoring or audit mechanisms, these oversights can lead to persistent vulnerabilities and external abuse.

B. Recommendations and Best Practices

Improving the security posture of local LLM frameworks requires coordinated efforts across tooling, usage, and community practices.

Tool developers: enforce secure defaults and restrict exposure. Developers of LLM-serving frameworks should adopt secure-by-default principles as a baseline. All interfaces should bind to the local loopback address unless explicitly configured otherwise, and sensitive endpoints involving model management, file access, or system inspection should require authentication or confirmation even in local testing environments. Debugging interfaces and metadata endpoints should remain disabled by default and only be exposed through deliberate configuration. In extensible platforms that support third-party modules or scripts, execution should be sandboxed, and plugin behavior must be scoped and controlled to prevent arbitrary access or privilege escalation. Documentation must emphasize the presence of security-critical endpoints, provide clear guidance for enabling authentication and authorization, and promote best practices during local or production deployment. Security features should be surfaced rather than buried as optional configurations.

System deployers: treat local interfaces as externally reachable. Operators deploying LLM systems must treat every interface, even those running locally, as potentially accessible

to untrusted users, especially in shared-host, containerized, or local network environments. Interfaces should be explicitly reviewed and minimized, unnecessary endpoints disabled, and authentication mechanisms activated wherever available. Networking configurations must be handled with caution. Container networking, port forwarding, and cross-device accessibility can expose services far beyond what is intended. Interfaces that handle prompt submission, queue monitoring, or tool invocation must be closely audited and protected against misuse or overuse. Regular security testing should be incorporated into deployment pipelines, including automated scans for exposed interfaces and simulated jailbreak attempts. Logging mechanisms should be comprehensive enough to support post-incident analysis and real-time anomaly detection.

Community stakeholders: promote secure deployment norms. As LLM frameworks transition rapidly from research prototypes to production-grade systems, the surrounding ecosystem must keep pace by establishing a culture of secure deployment. Insecure defaults in upstream tools often propagate downstream without scrutiny, increasing the systemic risk of widespread vulnerabilities. The community should actively develop and maintain secure deployment templates, configuration checklists, and audit frameworks tailored to LLM-serving use cases. Public repositories should annotate models and frameworks with deployment guidance, security metadata, and interface descriptions to help users apply safe configurations. More broadly, deployment security should be treated on par with performance and usability, supported by transparency, disclosure, and community-led testing.

C. Limitation

Deployment Framework Coverage. Our study focuses on a limited set of LLM deployment frameworks. We analyzed major platforms such as ComfyUI, Ollama, and Text Generation Web UI. Nevertheless, the insecure practices identified, including sensitive data exposure, unauthorized access, and resource abuse, reflect systemic issues common to LLM deployments. These risks are not limited to the studied frameworks and are broadly applicable across the ecosystem.

Sampling Coverage. Given the identification of over 320,000 exposed services, exhaustive probing would have imposed undue load on platform servers. To ensure ethical sampling while maintaining statistical rigor, we adopted a 95% confidence level with a 5% margin of error. Although representative, our sample may not capture certain rare or customized environments, which future work could further explore.

D. Ethical Considerations

All activities are carefully designed to minimize impact and respect the privacy of service operators. We probe only publicly accessible endpoints using non-destructive, read-only API requests, without attempting to bypass authentication, access private data, or exploit vulnerabilities. Sensitive findings are handled responsibly following coordinated vulnerability disclosure practices when appropriate. The study follows institutional and legal ethical guidelines for Internet research and collects no personally identifiable information.

VI. 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.

REFERENCES

[1] D. AI, "Deepseek llm," https://github.com/deepseek-ai, 2025.

[2] N. AI, "Gpt4all," https://github.com/nomic-ai/gpt4all, 2024.

[3] N. Alshahwan, J. Chheda, A. Finogenova, B. Gokkaya, M. Harman, I. Harper, A. Marginean, S. Sengupta, and E. Wang, "Automated unit test improvement using large language models at meta," in *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, ser. FSE 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 185–196. [Online]. Available: https://doi.org/10.1145/3663529.3663839

[4] Anonymous, "Awesome free ollama," https://freeollama.oneplus1.top, 2024.

[5] A. Bambhaniya, R. Raj, G. Jeong, S. Kundu, S. Srinivasan, M. Elavazhagan, M. Kumar, and T. Krishna, "Demystifying platform requirements for diverse llm inference use cases," *arXiv preprint arXiv:2406.01698*, 2024.

[6] G. Chen, Z. Qin, M. Yang, Y. Zhou, T. Fan, T. Du, and Z. Xu, "Unveiling the vulnerability of private fine-tuning in split-based frameworks for large language models: A bidirectionally enhanced attack," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 2904–2918. [Online]. Available: https://doi.org/10.1145/3658644.3690295

[7] L. Chen, Y. Wu, C. Wen, S. Wang, L. Zhang, B. Yu, Q. Sun, and C. Zhuo, "An agile framework for efficient llm accelerator development and model inference," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3676536.3676753

[8] J. Y. F. Chiang, S. Lee, J.-B. Huang, F. Huang, and Y. Chen, "Why are web ai agents more vulnerable than standalone llms? a security analysis," *arXiv preprint arXiv:2502.20383*, 2025.

[9] W. H. Community, "Fofa: Cyberspace asset search engine," https://fofa.info/, 2024.

[10] N. Contributors, "Nextchat," https://github.com/ChatGPTNextWeb, 2024.

[11] C. Developers, "Comfyui," https://github.com/comfyanonymous/ComfyUI, 2024.

[12] F. Firouzi, S. S. R. Nakkilla, C. Fu, S. Banerjee, J. Talukdar, and K. Chakrabarty, "Llm-aid: Leveraging large language models for rapid domain-specific accelerator development," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '24. New York, NY, USA: Association for Computing Machinery, 2025. [Online]. Available: https://doi.org/10.1145/3676536.3697135

[13] O. Foundation, "Owasp top 10 for llm applications 2025," https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/, 2024.

[14] G. Gerganov, "llama.cpp," https://github.com/ggerganov/llama.cpp, 2024.

[15] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection," in *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 79–90. [Online]. Available: https://doi.org/10.1145/3605764.3623985

[16] F. He, T. Zhu, D. Ye, B. Liu, W. Zhou, and P. S. Yu, "The emerged security and privacy of llm agent: A survey with case studies," *arXiv preprint arXiv:2407.19354*, 2024.

[17] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, Dec. 2024. [Online]. Available: https://doi.org/10.1145/3695988

[18] X. Hou, Y. Zhao, and H. Wang, "The next frontier of llm applications: Open ecosystems and hardware synergy," *arXiv preprint arXiv:2503.04596*, 2025.

[19] F. Huq, J. P. Bigham, and N. Martelaro, "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.

[20] A. Inc., "Ray serve," https://docs.ray.io/en/latest/serve/index.html, 2024.

[21] T. Isachenko and S. Bhuiyan, *Generative AI with local LLM*. Timur Isachenko, 2024.

[22] P. Jupyter, "Jupyter notebook," https://jupyter.org/, 2024.

[23] M. Lazuka, A. Anghel, and T. Parnell, "Llm-pilot: Characterize and optimize performance of your llm inference services," in *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2024, pp. 1–18.

[24] A. Li, Y. Zhou, V. C. Raghuram, T. Goldstein, and M. Goldblum, "Commercial llm agents are already vulnerable to simple yet dangerous attacks," *arXiv preprint arXiv:2502.08586*, 2025.

[25] B. Li, Y. Jiang, V. Gadepally, and D. Tiwari, "Llm inference serving: Survey of recent advances and opportunities," *arXiv preprint arXiv:2407.12391*, 2024.

[26] S. LLC, "Shodan: The search engine for internet-connected devices," https://www.shodan.io/, 2024.

[27] F. Mohammad Ali Pour and M. Rashidi, "Web llm attacks: Unveiling the future of cyber threats," *Available at SSRN 5049058*, 2024.

[28] NIST National Vulnerability Database, "Apache http server 2.4.29 vulnerabilities," https://nvd.nist.gov/vuln/detail/CVE-2019-0211, 2024.

[29] ——, "Nginx 1.14.0 vulnerabilities," https://nvd.nist.gov/vuln/detail/CVE-2019-20372, 2024.

[30] N. V. D. (NVD), "Cve-2024-37032: Ollama remote code execution vulnerability," https://nvd.nist.gov/vuln/detail/CVE-2024-37032, 2024.

[31] ——, "Cve-2024-6707: Openwebui arbitrary file upload vulnerability," https://nvd.nist.gov/vuln/detail/CVE-2024-6707, 2024.

[32] oobabooga, "Text generation web ui," https://github.com/oobabooga/text-generation-webui, 2024.

[33] OpenAI, "Introducing openai o3 and o4-mini," https://openai.com/index/introducing-o3-and-o4-mini, 2025.

[34] N. Pesati, "Security considerations for large language model use: Implementation research in securing llm-integrated applications," *Available at SSRN 4962370*, 2024.

[35] C. Project, "Llamafile," https://github.com/Mozilla-Ocho/llamafile, 2024.

[36] S. Ramírez, "Fastapi/swagger ui," https://fastapi.tiangolo.com/, 2024.

[37] M. Shetty, Y. Chen, G. Somashekar, M. Ma, Y. Simmhan, X. Zhang, J. Mace, D. Vandevoorde, P. Las-Casas, S. M. Gupta, S. Nath, C. Bansal, and S. Rajmohan, "Building ai agents for autonomous clouds: Challenges and design principles," in *Proceedings of the 2024 ACM Symposium on Cloud Computing*, ser. SoCC '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 99–110. [Online]. Available: https://doi.org/10.1145/3698038.3698525

[38] A. Team, "Anythingllm," https://github.com/Mintplex-Labs/anything-llm, 2024.

[39] G. Team, "Gradio," https://www.gradio.app/, 2024.

[40] J. Team, "Jan," https://github.com/janhq/jan, 2024.

[41] K. . Team, "Zoomeye: Cyberspace search engine," https://www.zoomeye.org/, 2024.

[42] O. Team, "Ollama," https://ollama.com, 2024.

[43] O. W. Team, "Open webui," https://github.com/open-webui/open-webui, 2024.

[44] vLLM Team, "vllm," https://github.com/vllm-project/vllm, 2024.

[45] F. Wu, S. Wu, Y. Cao, and C. Xiao, "Wipi: A new web threat for llm-driven web agents," *arXiv preprint arXiv:2402.16965*, 2024.

[46] F. Wu, N. Zhang, S. Jha, P. McDaniel, and C. Xiao, "A new era in llm security: Exploring security concerns in real-world llm-based systems," *arXiv preprint arXiv:2402.18649*, 2024.

[47] G. Wu, Z. Zhang, Y. Zhang, W. Wang, J. Niu, Y. Wu, and Y. Zhang, "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*, 2025.

[48] Q. Xie, D. Li, M. Xiao, Z. Jiang, R. Xiang, X. Zhang, Z. Chen, Y. He, W. Han, Y. Yang *et al.*, "Open-finllms: Open multimodal large language models for financial applications," *arXiv preprint arXiv:2408.11878*, 2024.

[49] H. Yao, H. Shi, Y. Chen, Y. Jiang, C. Wang, Z. Qin, K. Ren, and C. Chen, "Controlnet: A firewall for rag-based llm system," *arXiv preprint arXiv:2504.09593*, 2025.

[50] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang, "A survey on large language model (llm) security and privacy: The good, the bad, and the ugly," *High-Confidence Computing*, p. 100211, 2024.