Rising Fast, Prone to Risk: How Open-Source LLM-Powered Apps Are Designed and Secured

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

Alvaro Vazquez
Department of Computer Science
Texas A&M University-Corpus Christi
Corpus Christi, USA
avazquez16@islander.tamucc.edu

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

Kadir Alpaslan Demir
Department of Computer Science
Texas A&M University-Corpus Christi
Corpus Christi, USA
kadiralpaslan.demir@tamucc.edu

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

***Abstract*—The rapid rise of large language models (LLMs) has driven their widespread adoption, especially as the core component of open-source applications, which we refer to as LLM-Powered Apps (LPAs). Despite the rapid growth of this ecosystem, little is known about how these applications are built in the open-source world, especially in terms of their architectural and design decisions, deployment strategies, and security practices, which remain poorly understood.**

In this paper, we conduct a comprehensive empirical study of 89 popular open-source LPAs on GitHub, with the goal of characterizing their design choices and identifying common security and safety concerns. We systematically collect a set of architectural and operational attributes, classify each LPA by its primary purpose and analyze how functionality influences architectural and security design. Our findings reveal dominant design patterns as well as recurring risks, such as inadequate access control, lack of telemetry transparency, and design assumptions that break down in complex runtime environment. By surfacing these trends and vulnerabilities, our study provides a foundational understanding of how LPAs are currently built and deployed in the open-source ecosystem. The results offer practical insights for developers, researchers, and platform maintainers seeking to build more robust and secure LLM-integrated software systems.

***Index Terms*—LLM-Powered App, Architectural Decision, Design Decision, Security and Safety Concern**

I. Introduction

The integration of large language models (LLMs) into software as the core component that enables the primary functionality has become increasingly popular [1]. We call such applications LLM-Powered Apps (LPAs). This trend has given rise to enormous projects on open-source platforms [2]–[4] (*e.g.*, GitHub), where developers collaborate and share code to leverage and explore the potential of LLMs in a wide range of domains, from clinical decision support [5], [6] to code generation [7], [8].

As the trend continues, substantial research efforts have been devoted to understanding, identifying and addressing challenges introduced by LLM and its integration. One key focus is to detect and prevent security and safety issues introduced by adopting LLMs into software [9]–[13]. Another primary direction is to understand the difficulties and challenges of developing LLM-integrated software [14], [15], where several empirical studies and surveys have been conducted on questions posted by developers on Stack Overflow [15], OpenAI developer forum [14], [15], or collected through interviews with developers [16]. Although their insights provides valuable perspectives on user confusion and high-level development difficulties, they often overlook the concrete, real-world issues that arise during the design and development of open-source LPAs. Moreover, a systematic understanding of how LLMs are architected into real-world applications, especially in the open-source ecosystem, remains limited. Specifically, little is known about the architectural and design decisions, deployment strategies, and security or privacy considerations in these LPA implementations. This lack of visibility limits our ability to assess the robustness, secure and safe deployability of such systems in practice.

To address this gap, we conduct a comprehensive empirical study of popular open-source LPAs, focusing on projects with over 1,000 GitHub stars. Our goal is to characterize their architectural and design choices and uncover common patterns, trends, and potential risks. We collect a diverse set of technical and operational attributes, including programming languages, LLM access modes (*e.g.*, remote vs. local), deployment environments, authentication and access control mechanisms, caching and logging strategies, and user-facing features such as file input. These attributes are selected to reflect key dimensions of software architecture [17]–[19], *e.g.*, component choice, system interaction, deployment context, and cross-cutting concerns, as well as their implications for security and safety.

Our study reveals several notable findings. First, we iden-

tify dominant architectural trends in LPA design, including common technology stacks, model integration methods, and platform choices. Second, and more critically, we observe a range of recurring security and safety risks in open-source LPAs, such as the absence or ineffective implementation of access control, poor handling of user inputs and logs, and assumptions that break down when LPAs are deployed in complex environments (*e.g.*, cloud). These issues highlight a security gap between design and deployment, suggesting that many LPAs may not be ready for use in production or multi-user contexts without significant hardening.

By providing the first architectural and security-focused analysis of open-source LPAs at scale, our study aims to inform developers, researchers, and platform providers about current practices and common pitfalls. The results not only serve as a snapshot of the state of LPA development, but also as a foundation for future tools, guidelines, and frameworks to support more robust, secure, and trustworthy LLM-powered software. The dataset, results, and source code required to reproduce our study findings are publicly available [20].

II. Definition and Related Work

A. Definition of LPAs

The term "LLM-integrated application" (*a.k.a.,* "LLM-integrated system" and "LLM-based application") has been used in previous research [21]–[24] to describe software systems where LLMs are incorporated as an auxiliary component or enhancement. These applications often leverage LLMs for certain tasks within broader systems that could still function without them [25], [26]. This study focuses on a different category formed recently: LLM-Powered Applications (LPAs), which represents a new class of applications that emerged as LLMs became more advanced, accessible and practical for everyday use. We define an LPA as an application in which the LLM acts as the central component, directly enabling the application's primary functionality. In LPAs, the system logic, user interaction, or service output fundamentally depends on the LLM. Without the LLM, the application becomes functionally ineffective or entirely non-operational.

B. Studies on LLM-Integrated Software

As LLMs have been largely integrated into software, numerous research efforts focus on the security risks that arise from this trend. Sallou et al. [9] highlighted threats to the validity of LLM-based research and propose guidelines for software engineering researchers and LLM providers. Priyanshu et al. [27] discovered that ChatGPT retains a large portion of personally identifiable information during inference, exposing concerns about privacy-related policies in LLMs. Yao et al. [28] and Jin et al. [29] conducted literature reviews on LLMs, identifying that LLMs pose risks for user-level attacks due to their advanced reasoning abilities.

Researchers also investigate how software developers use LLMs to improve their efficiency. Tufano et al. [30] categorized the types of tasks that are automated via ChatGPT by developers, which provides the insights about common usage scenarios of LLMs in software development. Chouchen et al. [31] presented a taxonomy to categorize the various topics that developers discuss with ChatGPT in order to understand and improve the interaction between developers and LLMs.

Moreover, with the evolution of LLMs, many studies have been conducted to explore the challenges and difficulties in developing LLM-integrated systems. Zhao et al. [32] analyzed the current landscape of LLM from app stores and provided future research directions to foster innovation and collaboration among stakeholders. Weber [21] investigated the use of LLMs as software components and classified how LLMs are integrated into software applications. Yang et al. [33] investigated GitHub issues from open-source AI repositories to understand developers' problems while employing AI systems. Similarly, Chen et al. [14] conducted an empirical study by analyzing questions posted by LLM developers on OpenAI developer forum [34] and developed a taxonomy of the challenges they faced. Nahar et al. [16] identified challenges in developing and evaluating LPAs through interviews with software developers from industry and propose potential solutions to improve software quality. Abeysinghe and Circi [35] also addressed the difficulties in evaluating LPAs. To complement existing work in this area, we investigate open-source LPAs to explore the landscape of LPA as well as observed and potential security and safety issues omitted by open-source LPA developers.

III. Methodology and Study Design

A. Research Questions

To understand the architectural and design decisions, security and safety considerations of open-source LPAs, our study aims to answer the following research questions (RQs):

- ***RQ1*** *What are the dominant architectural and design decisions in popular open-source LPAs?* This question investigates architectural and design decisions such as programming language, platform support, LLM access mode (remote vs. local), deployment environment (cloud/on-premises), and presence of caching or logging mechanisms. These attributes reflect the underlying structure and runtime behavior of LPAs.
- ***RQ2*** *How are user access and security handled in open-source LPAs?* We examine the use of authentication mechanisms, access control policies (*e.g.*, role-based), personal file upload handling, and logging of user interactions. These features reveal how LPAs handle potentially sensitive inputs and outputs and enforce trust boundaries.
- ***RQ3*** *What purposes do open-source LPAs serve, and how do their purposes correlate with architectural and security design choices?* By categorizing LPAs by their primary function, we explore whether the intended use case influences architectural decisions or security posture, for example, whether RAG tools more often support file uploads, or assistants rely on different authentication models.

B. Data Collection

To answer our RQs, we conducted a multi-phase data collection process involving both automated and manual steps to

TABLE I
DATA ATTRIBUTES COLLECTED FOR EACH LPAS IN OUR STUDY.

RQ	Attribute	Description
RQ1	Used LLMs & Access Mode	Which LLM(s) are integrated into the LPA and whether they are accessed via a remote API or run locally.
	Platform Support	Which platform the LPA runs (*e.g.*, web browser, desktop, cross-platform).
	Deployment Environment	Where the LPA is deployed: on-premises, cloud or hybrid?
	Caching Mechanism	Whether the LPA uses caching (*e.g.*, response caching, model output caching) to improve efficiency.
	Logging Mechanism	Whether the LPA logs outputs that contain user queries and answers from LLMs.
	Primary Programming Language	The main language(s) (*i.e.*, whose percentage is greater than 10%) used to implement the LPA .
RQ2	User Access Model	Who (or how many users) the LPA is designed for.
	Authentication Mechanism	Whether the app has login or authentication, and what kind (*e.g.*, none, basic auth, OAuth) is adopted.
	Access Control Policy	Presence and enforcement of role-based or group-based access control.
	Telemetry	Whether the LPA logs or collects user interactions, queries, or outputs for analytics or other purposes.
	Personal File Input Support	Whether the LPA allows users to upload personal files for processing (*e.g.*, PDF).
RQ3	Information Retrieval	Whether the LPA retrieves external documents or content that commonly associated with RAG systems.
	Autonomous Task Execution	Whether the LPA includes multi-step reasoning or autonomous workflows beyond single-turn prompts.
	Code-Related Functionality	Whether the LPA is focused on supporting software development tasks (*e.g.*, code generation, refactoring).
	Structured Output Generation	Whether the LPA automates the creation of documents with fixed structure or format (*e.g.*, resumes).
	Multi-LLM or API Abstraction	Whether the application abstracts access to multiple LLMs or providers through a unified interface.
	Productivity Support Features	Whether the app includes note-taking, calendar integration, or personal workflow features.

ensure relevance, completeness, and quality. To collect the data attributes required in our study, we chose GitHub as the target platform because GitHub's extensive ecosystem provides a rich source of data, on which many existing studies [30], [33], [36], [37] are conducted, making it an ideal platform for our investigation on open-source LPAs.

1) Collection of LPA Repositories: We first identified candidate LPA repositories on GitHub using the search API [38] that queries for projects using LLM-related keywords as topics. The keywords contain "llm", well-known LLMs (*e.g.*, "chatgpt") and their companies (*e.g.*, "openai"), as well as LLM-powered software (*e.g.*, "chatgpt-app"). To focus on influential applications, we included only repositories with over 1,000 GitHub stars. In our search, we noted that many repositories are tagged with multiple LLM-related keywords, and we report each repository only once. This initial search yielded 312 repositories.

Next, we applied a manual process to filter out repositories that were not aligned with the scope of LPAs. The filtering process was guided by our definition of LPAs. This refinement excludes 122 repositories that include 22 infrastructure components used to build, optimize and run LLMs, 20 tutorials, 19 curated lists of LLM-related techniques, 9 containing the keyword but are irrelevant to the study, and 1 being purely documentation. Another 52 repositories were excluded due to non-English documentation. Finally, a dataset of 89 open-source LPA repositories were selected for our study.

Note that our study includes only repositories with 1,000 or more GitHub stars, ensuring visibility but potentially biasing results by excluding newer or less-promoted projects.

2) Collection of Data Attributes: To answer our RQs, we collected a set of descriptive attributes from each LPA repository to capture their design and development features. Table I summarizes the data attributes we collected for each LPA. We extract these attributes through an automated scrapper of repository documentation, configuration files and source code with manual inspection. The collected data form the basis for understanding the characteristics of LPAs in OSS ecosystem.

C. Data Analysis

To answer the RQs, we use a combination of quantitative and qualitative methods on the data attributes shown in Table I. We begin by identifying trends in the architectural and design decisions of LPAs through frequency analysis of core implementation attributes such as programming languages, LLM access modes, deployment environments, and platform support. We then examine security-relevant features, including authentication mechanisms, access control policies, personal file input support, and logging behaviors, to assess how these applications handle user access and sensitive data. To explore potential relationships between application purpose and design choices, we compare the distributions of architectural and security features across different categories of LPAs. Finally, we conduct a qualitative review of patterns and edge cases that reveal potential risks, such as security gaps between design intent and actual enforcement, or issues that may arise when LPAs are deployed in complex or shared environments like cloud platforms. This approach allows us to extract both broad trends and nuanced insights into the current state of open-source LPA development.

IV. STUDY RESULTS

A. The dominant architectural and design decisions (RQ1)

Used LLMs & Access Mode The 89 LPAs support integration with 159 different LLMs [1] in total. The four most frequently supported models are OpenAI's ChatGPT (71 out of 89), Llama (56), Google's Gemini (43) and Claude (40), underscoring the wide adoption of OpenAI's products. While 23 LPAs support only a single LLM integration, the majority are designed for multi-LLM integration.

Our study reveals two main approaches to access LLMs: through remote API services or LLM inference frameworks

[1]This includes the LLMs explicitly listed in the LPA's documentation.

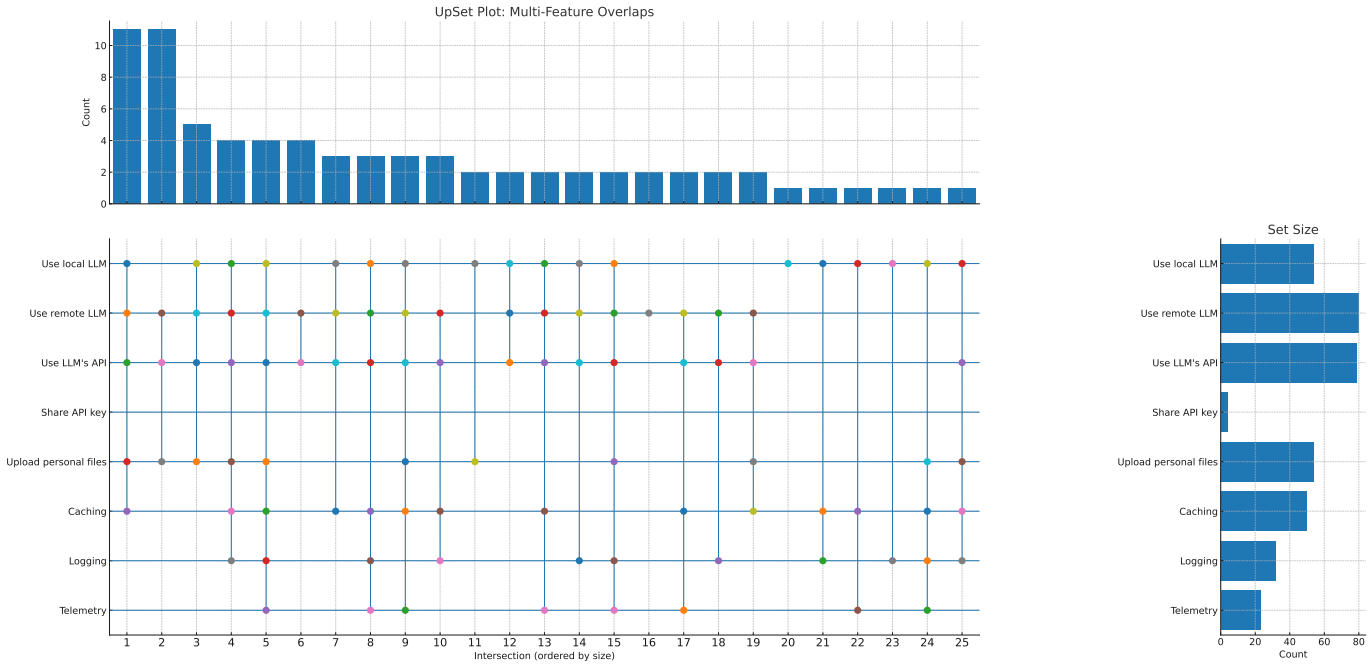

Fig. 1. Overlaps in Architectural and Security Attributes

(LIFs) [2] for running LLM locally. Among the 89 LPAs, 79 (88.76%) rely on LLM API keys, reflecting the popularity of API-based integration, especially for services built on top of foundational LLM providers such as OpenAI. However, only 55 of them explicitly encourage users to store their keys as environment variables in their documentation (*e.g.*, README). Of the 79 API-based LPAs, 4 are explicitly designed for team use through shared API keys managed under role-based access controls. For 12 others, it is unclear whether keys are shared on their cloud service, as they are deployed as closed-source platforms with proprietary API interfaces and subscription models, making internal handling of keys opaque.

8 LPAs in our dataset only support local LLMs, while 46 support both local and remote access. Among the 54 LPAs that enable local LLM integration, the most commonly used LIF is Ollama (38), followed by llama.cpp (7), LLM Studio (7) and LocalAI (4). The growing adoption of local inference tools demonstrates their increasing practicality and appeal for developers by offering greater controls and enhancing privacy for end users. This highlights a trend among LPAs to support hybrid access, allowing users to choose between cloud-based APIs and local inference based on their privacy needs.

Platform Support The 89 LPAs in our study offer support across 13 different platforms. MacOS and Windows are tied as the most supported platforms (each supported by 32 LPAs), followed by Linux with 29. All LPAs adopt cross-platform design: 38 offer platform-specific downloads, and the most common combination is MacOS, Windows and Linux (supported by 28). The rest LPAs use general installation (*e.g.*, `pip` or `go`) or platform-independent way (*e.g.*, Docker and web browsers) to achieve compatibility. Moreover, 10 LPAs are also expanding to mobile platforms such as iOS, WatchOS and Android, reflecting a new trend of adapting LPAs for lower-powered computing environments.

Deployment Environment 86 out of the 89 LPAs support local deployment to personal devices (*e.g.*, PCs, smartphones or private servers), commonly referred as "on-premises". 5 LPAs explicitly support deployment to cloud platforms such as AWS, Google Cloud Platform and Azure, with 1 of them supporting hybrid deployment. Moreover, 22 LPAs adopt Docker for packaging, enabling LPAs originally designed for on-premises deployment to be easily deployed to cloud with minimum deployment overhead and compatibility issues.

Caching & Logging Mechanisms To enhance performance and efficiency, 50 LPAs implement caching mechanisms (*e.g.*, through `.langchain.db`, `.json` or Redis [39]), enabling faster responses by reusing previous query results. Besides, 32 incorporate logging of user queries and/or LLM responses for debugging and monitoring. Surprisingly, none of the LPAs uses hardcoded paths to store cache files. This suggests the awareness of potential data leak risks and a deliberate effort to enhance security through careful handling of data storage.

Primary Programming Language TypeScript emerges as the mostly used programming language, appearing in 42 out of the 89 selected LPAs (47.19%). This highlights TypeScript's stronghold in building interactive web-based apps, favored for its type safety and modern development features. Python, in second place, is adopted by 30 LPAs, reflecting its widespread adoption in AI-driven development. Among the 89 LPAs, 64 are written primarily in one programming language, with 90% of the codebase excluding configuration files and front-end

[2]Commonly used to run LLMs locally, such as Ollama and LM Studio.

TABLE II
PURPOSE CATEGORIES OF LPAS IN OUR STUDY.

Category	Associated Attribute(s)	Example Use Cases	#LPAs (%)
Developer Tool	Code-Related Functionality	AI code assistants, copilot tools, scrapers	25 (28.09%)
RAG Tool	Information Retrieval	Document QA, chat over PDFs	12 (13.48%)
Task Automation Agent	Structured Output Generation	Resume builders, job application generators	10 (11.24%)
LLM Gateway	Multi-LLM or API Abstraction	Middleware APIs for OpenAI, Claude	5 (5.62%)
Autonomous RAG Agent	Information Retrieval, Autonomous Task Execution	Internet-answering AI agents	4 (4.49%)
Productivity Assistant	Productivity Support Features	Meeting assistants	2 (2.25%)

code (*e.g.*, HTML and CSS). The remaining 25 LPAs are written primarily in two or more programming languages, where the most popular combination is Python and TypeScript (by 9 LPAs), indicating their pivotal role in creating user-friendly interfaces and integrating LLM functionalities seamlessly.

B. User Access & Security (RQ2)

User Access Model We classified user access models into three categories: (1) single-user mode, (2) multi-user mode, where multiple users share resources via the cloud, and (3) team-based mode, where resources are shared with role-based access control. All 89 LPAs analyzed support single-user mode. Among them, 52 support only single-user mode, 21 support both single-user and multi-user modes, and 10 support all three models. In total, 37 LPAs support multi-user mode (including 7 that provide publicly accessible demo websites), and 16 LPAs support team-based mode. The increasing adoption of multi-user and team-based access models as well as on-premises deployment suggests that LPAs are evolving beyond personal use cases toward collaborative environments and production-level applications. This trend highlights a growing demand for shared resources, streamlined deployment and proper access control policies for resource-sharing LPAs.

Authentication Mechanism A wide range of authentication methods are observed across our dataset. The most common method is to use user's personal LLM API keys, implemented by 22 (24.72%) LPAs. Subscription-based authentication for cloud services follows with 14 LPAs, 7 of which support their own API keys. Other mechanisms include single sign-on (SSO, by 4 LPAs) and the use of passwords (by 6 LPAs), sometimes combined with alternative methods such as Magic Links and Google OAuth. More secure methods such as two-factor authentication (2FA) are less common, observed in only one LPA, indicating that most LPAs still rely on simple and less robust methods for authentication.

Notably, 5 LPAs using local LLMs and 15 LPAs supporting both local and remote LLM access implement no form of access control, which are designed for simple runtime environment (*e.g.*, personal devices). More importantly, 2 of these applications support multi-user mode, which further amplifies the potential security risks due to the lack of user isolation and permission enforcement.

Access Control Policy Among the 15 LPAs that support team-based mode, 9 adopt role-based access control [40] (RBAC) and one adopts Lightweight Directory Access Protocol [41] (LDAP) for directory-based access control, incorporating password-based authentication. Besides, 5 LLM development frameworks provide templates for implementing RBAC in LPAs, underscoring its relevance in organizational settings. This reflects developer's effort to enable fine-grained permission control, essential for collaborative environment.

Telemetry Telemetry plays a critical role for both developers and users, however, which is only presented in 24 of our dataset. For developers, telemetry provides insights into usage patterns and system behavior, which are essential for performance optimization. From user's perspective, telemetry offers transparency into the data being collected, enabling users to evaluate the privacy practices of an LPA and decide whether it aligns with their data sensitivity and usage preferences.

Personal File Input Support A total of 54 LPAs allow users to upload personal files for processing by LLMs. For example, RAG Web UI [42] accepts PDF, DOCX, MD and TXT. This widespread support for personal file input highlights a strong emphasis on user-specific context and customization, suggesting that many LPAs are being designed not just for general querying, but for deeply personalized workflows such as document analysis, academic research, and domain-specific retrieval-augmented generation.

C. Overall Attribute Overlaps

Figure 1 presents an UpSet diagram illustrating how key architectural and security-related attributes co-occur across the analyzed open-source LPAs. The left-bottom plot encodes each intersection as filled dots (with a connecting vertical line) indicating which attributes co-occur in that intersection (ordered left-to-right by frequency). The top bar chart shows the intersection sizes aligned to the bottom plot columns, enabling immediate comparison of how common each specific combination is. On the right-hand side, a horizontal bar chart reports the marginal size for each individual attribute. These three coordinated views summarize both overall prevalence and the structure of co-occurrence patterns, highlighting which attribute combinations are most common in the dataset.

D. Purposes of LPAs (RQ3)

To understand the functional diversity of open-source LPAs, we classify each project in our dataset based on its primary purpose and attributes shown in Table I. Table II presents six purpose categories, together with how these categories relate to the concrete attributes observed in the LPAs, ordered by the number of projects observed. The most common are *Developer Tools*, leveraging LLMs to assist in software

engineering tasks such as code generation or debugging. *RAG Tools* use retrieval-augmented generation to provide document-grounded responses, while *Task Automation Agents* employ LLMs to generate structured outputs such as resumes or job applications. We also identify more complex categories such as *Autonomous RAG Agents*, which combine retrieval with autonomous reasoning for multi-step task completion or research, and *LLM Gateways*, which offer a unified access layer to multiple model providers. Finally, *Productivity Assistants* focus on enhancing daily workflows like note-taking and calendar planning through LLM integration.

By categorizing LPAs in this way, we enable a comparative analysis of how an LPA's purpose may shape its architectural and security design choices, discussed in Section VI-A.

V. Security & Safety Issues in LPAs

While analyzing the open-source LPAs, we identified a range of recurring security and safety issues that stem from two aspects, software architecture and access controls.

A. Software Architecture

Migrate to Complex Runtime Environment All LPAs in our study are primarily designed for single-user mode on personal devices, such as smartphones and personal computers, with relatively simple runtime environment. However, installing and deploying these apps in complex runtime environments (*e.g.*, Google Cloud or AWS), where resources are shared among multiple users, can expose security flaws if the default app configurations do not support such complex environment. Such a migration can introduce significant security challenges that are notoriously difficult to test and debug due to the complexity of cloud systems.

One example is ChuanhuChatGPT [43], which uses `config.json` to store API keys and follows the best practices. However, the app is designed for multi-user mode with their chat histories stored locally [44], which makes it possible to locate other users' chat histories and manually see other user's conversations if the app is installed on a server shared by multiple users.

Alternative Ways for Using LLMs In addition to REST APIs and LIFs, we observe a third approach to utilizing LLM services: wrapping websites as desktop applications. For example, ChatGPT Desktop Application [45] repackages the ChatGPT website into a desktop app using Microsoft Edge WebView2 [46], requiring users to log in via the official ChatGPT web portal. Similarly, Pake [47] functions as a general-purpose wrapper for various websites—including Twitter, YouTube, and ChatGPT—transforming them into native-like desktop experiences. Another example, Chat AI Desktop [48], wraps the ChatGPT website but also offers support for API-based access, blending web and programmatic interactions.

The potential security issues lie in the third-party components used by developing an app [49], which can make calls to security-sensitive APIs. In our study, this refers to Microsoft Edge WebView2 for the above mentioned repositories, posing potential security risks. Microsoft Edge WebView2 has been associated with numerous security vulnerabilities, as documented in the Microsoft Release Notes [50], with many of them being exploits observed in real-world scenarios when using Chromium-based Microsoft Edge. Developers must remain vigilant in monitoring and addressing these vulnerabilities to ensure the overall security of their apps. Meanwhile, they should regularly update their dependencies to mitigate the risk of exploitation and incorporate robust security measures into their development practices.

B. Access Control Issues

Access Control Policy In our study, 4 LPAs are designed to share ChatGPT resources in order to reduce usage costs. One example is the TypeScript implementation of GPT4Free [51]. However, the remaining three LPAs either lack adequate access control or implement potentially invasive policies, raising serious privacy and security concerns.

ChatGPT Web Share [52] shares a single ChatGPT Plus account with multiple users, and implements a simple RBAC to manage users [53]. However, the administrators can view the conversation history that users are using and have deleted [54]. Allowing administrators of an app to view conversation history that users have deleted raises significant safety and privacy concerns, including violations of user trust, exposure of sensitive information, and risks of unauthorized access. This practice can lead to potential data breaches, internal misuse, and legal issues due to non-compliance with data protection regulations like GDPR [55]. Meanwhile, it raises ethical concerns by undermining user autonomy and transparency, as users expect their deleted data to be permanently removed and not accessible by administrators. The same condition also applies to Chat2DB [56] and AWS GenAI LLM Chatbot [57]. These LPAs have insufficient access control policies, which can result in vulnerabilities such as conflicted authentication, permission escalation and RBAC failures. This highlights the importance of correctly designing and thoroughly testing access control policies and configurations to prevent security flaws and ensure that user permissions are managed effectively.

Cache and Log In our study, 82 out of 89 repositories have their own caching and/or logging mechanisms to remember queries and prompts for faster performance or debugging. Surprisingly, some of these apps have no authentication methods at all, which highlights a critical gap in security practices.

As we mentioned previously, all the LPAs in our study are designed for single-user scenario, which assumes that the runtime environment is safe and only the assigned user can use this app. This assumption makes the app design lack proper authentication and authorization, especially for those apps that use third-party caches and logs. For example, ChatGPT Web Share uses MongoDB [58] to store users' conversations, however, the MongoDB connection string `mongodb://cws:password@mongo:27017` is stored at `backend/config_templates/config.yaml`. There is no environment variables used for `cws` and `password`, which means they will be hard-coded into the string. This configuration makes users' information unsafe, because (1)

including the credentials directly in the connection string exposes them in plain text, which can be vulnerable if the connection string is logged, stored in source code, or shared; (2) the credentials are hard-coded in the app's configuration file, which can lead to accidental exposure, especially when the repository is public with hundreds of forks; (3) the connection string uses the `mongodb` protocol, which does not enforce encryption, meaning data transmitted between the client and the server is not encrypted, making it vulnerable to interception and man-in-the-middle attacks.

This concern also applies to standalone apps that are self-contained with an LLM, which could become more prevalent as compilers and runtimes evolve to be more powerful for supporting ML techniques. Developers must prepare for the future to prevent potential security issues arising from increasingly complex app architectures using third-party libraries.

Telemetry Only 24 LPAs in our study (27.27%) have explicit telemetry statements, indicating that they log user interactions, queries, or model outputs. In many cases, these statements are limited to brief mentions in documentation or README files, without specifying the scope, retention policy, or handling of logged data. The remaining applications either do not log user activity or do not disclose whether logging occurs—raising concerns about transparency and potential data misuse. Among those that do implement telemetry, few offer users meaningful control over logging behavior (*e.g.*, opt-out options), and even fewer provide detailed disclosures about how logs are used (*e.g.*, for analytics, debugging, or model improvement). Given that LLMs often process sensitive or personal information, the lack of clear and consistent telemetry practices may pose privacy risks, especially when LPAs are deployed in shared or multi-user environments. This highlights a broader issue: many open-source LPAs operate without strong data governance mechanisms, despite their potential to collect or expose user-generated content.

VI. Discussion

A. How Purpose Shapes Architecture and Security?

Our analysis shows that an LPA's functional purpose significantly shapes its architectural and security-related decisions. Among *Developer Tools*, over 85% support both local and remote LLMs, and nearly all require an API key for LLM access. However, only a minority (less than 20%) implement any access control mechanism such as RBAC or password login, even when offering cloud deployment. Caching is widely adopted in this category (*e.g.*, 75%), likely for performance optimization in iterative code generation or chat tasks. *Developer tools* also lead in telemetry support, with 9 out of 25 projects explicitly documenting telemetry collection—though opt-out controls are still rare.

In contrast, *Autonomous RAG Agents* present a more mixed picture. All agents in this category operate remotely or in hybrid modes, and 100% support personal file uploads to augment LLM reasoning. However, none of them implement caching for performance, which may limit efficiency in multi-step pipelines. Only one project enforces access control using OAuth method, despite the fact that at least two support multi-user configurations. These gaps indicate a substantial security risk, especially when these agents are deployed beyond personal devices.

RAG Tools typically function as document-aware assistants. These applications are more likely than others to include document upload support (75%) and developer-facing APIs (nearly 50%). Around half implement caching, and logging is fairly common, but access control is rarely enforced, even when cloud deployment is supported. Only a few tools offer explicit telemetry or permission management disclosures.

For other categories, we observe consistent reliance on remote LLMs and frequent exposure of REST APIs. While all applications in this category require API keys, shared keys are common, and robust user-level authentication or isolation is largely absent. Logging is minimal, and telemetry is rare, with only one interfaces disclosing data collection practices.

Across all categories, telemetry remains uncommon, with only 15% of LPAs providing explicit telemetry statements. Caching is prevalent in developer-oriented apps but nearly absent in agent-style or assistant-style tools. Access control mechanisms are implemented in fewer than 25% of projects overall, despite a growing number of LPAs supporting multi-user or cloud-based deployment.

These findings suggest that while the architectural complexity of LPAs is increasing, driven by purpose-specific needs, security and privacy practices are not keeping pace. Developers often optimize for functionality and compatibility, but overlook protections such as authentication, logging transparency, or data governance. This raises concerns as open-source LPAs become widely adopted in collaborative settings.

B. What new security requirements arise from evolving LPAs?

The trend of multi-user and cloud deployment in LPAs, which always shares LLM resources, requires robust and flexible access control mechanisms to support well-defined permissions at user and/or resource level. The observed adoption of access control policies (*i.e.*, RBAC and LDAP) in our dataset indicates a good start for developers to focus on user data safety. This also suggests that while access control is beginning to evolve, it still lags behind authentication in terms of consistency and depth. The current landscape indicates that most LPAs are still in the early stages of supporting shared environments, but the adoption of RBAC and LDAP points to a shift toward more mature, team-oriented usage models.

Meanwhile, LPAs often consist of components and modules from third parties, which can create security gaps between different components, making the system vulnerable to malicious exploits. For example, inconsistent access controls can allow unauthorized users to gain access to other user's data [59]. Ensuring a unified and comprehensive access control policy is essential to mitigate these risks and to protect the integrity and confidentiality of the system as a whole.

Sharing LLM resources to save costs often face challenges due to undefined, insufficient or invasive access control policies. Properly designing and thoroughly testing access control

policies and configurations is crucial, especially for LPAs that store and analyze users' personal data, to prevent security vulnerabilities and ensure effective permission management. Additionally, caching and logging mechanisms in these apps must be secured with appropriate authentication and authorization to prevent data leaks and maintain data integrity.

C. What risks come with deploying LPAs to the cloud?

Most LPAs are designed for single-user mode with event-driven architecture and minimum access controls, which have limited adoption of threads or other complex architecture designs. Hence, running these LPAs on personal devices seems secure with no reported security issues. However, running them in a complex runtime environment (*e.g.*, cloud server shared by multiple users) presents significant challenges that demand careful redesigns. Otherwise, users on the same server may be able to access each other's files, resulting in data exposure [60]. This also emphasizes the importance of designing comprehensive access control mechanisms for LPAs.

The analysis includes only repositories with 1,000 or more GitHub stars. While this ensures visibility, it likely excludes newer or less-marketed projects that may have different architectural or security characteristics.

VII. Conclusion & Future Work

Our study sheds light on the architecture, design, and security aspects of LPAs. Our findings highlight the importance of prioritizing security measures and good practices, both for developers crafting LPAs and for users seeking to utilize LPAs. Our work helps foster a more secure and resilient ecosystem for LLM-powered software development.

First, based on the data collected in this study, we could develop concrete guidelines and explore the role of the technologies used as well as the characteristics of open-source development, while identifying best practices to create more resilient and secure LLM-powered applications. Second, we may create metrics for LPAs to evaluate the resilience and security of these applications. Third, future work could incorporate statistical analyses, such as testing correlations between app categories and security features or conducting regression analyses, to move beyond frequency counts and distributions and provide stronger evidence for relationships between app characteristics and architectural choices. Based on these best practices and metrics, we will be able to create frameworks, architecture and design best practices for creating LPAs that will contribute significantly to the LLM-powered application ecosystem. We envision that these types of applications will be a significant portion of software applications in the future. Therefore, this study and other empirical and conceptual studies will help build the foundation for a successful LLM-powered application ecosystem.

Acknowledgment

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

References

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

[2] "shubhamsaboo/awesome-llm-apps: Collection of awesome LLM apps with AI Agents and RAG using OpenAI, Anthropic, Gemini and opensource models. - github.com," https://github.com/Shubhamsaboo/awesome-llm-apps, [Accessed 03-04-2025].

[3] "Hannibal046/Awesome-LLM: Awesome-LLM: a curated list of Large Language Model - github.com," https://github.com/Hannibal046/Awesome-LLM, [Accessed 25-04-2025].

[4] "Models - Hugging Face - huggingface.co," https://huggingface.co/models?other=LLM, [Accessed 25-05-2024].

[5] P. Hager, F. Jungmann, R. Holland, K. Bhagat, I. Hubrecht, M. Knauer, J. Vielhauer, M. Makowski, R. Braren, G. Kaissis, and D. Rueckert, "Evaluation and mitigation of the limitations of large language models in clinical decision-making," *Nature medicine*, vol. 30, no. 9, pp. 2613–2622, 2024.

[6] B. Li, T. Meng, X. Shi, J. Zhai, and T. Ruan, "Meddm: Llm-executable clinical guidance tree for clinical decision-making," *arXiv preprint arXiv:2312.02441*, 2023.

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

[8] R. Chew, J. Bollenbacher, M. Wenger, J. Speer, and A. Kim, "Llm-assisted content analysis: Using large language models to support deductive coding," *arXiv preprint arXiv:2306.14924*, 2023.

[9] J. Sallou, T. Durieux, and A. Panichella, "Breaking the silence: the threats of using llms in software engineering," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, 2024, pp. 102–106.

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

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

[12] "Scalable extraction of training data from (production) language models," 2023.

[13] "A new era in llm security: Exploring security concerns in real-world llm-based systems," 2024.

[14] X. Chen, C. Gao, C. Chen, G. Zhang, and Y. Liu, "An empirical study on challenges for llm application developers," *ACM Trans. Softw. Eng. Methodol.*, Jan. 2025, just Accepted. [Online]. Available: https://doi.org/10.1145/3715007

[15] K. Alam, K. Mittal, B. Roy, and C. Roy, "Developer challenges on large language models: A study of stack overflow and openai developer forum posts," *arXiv preprint arXiv:2411.10873*, 2024.

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

[17] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley Professional, 2012.

[18] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 291–300.

[19] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," *Advances in neural information processing systems*, vol. 28, 2015.

[20] "apace-lab/LLM-Powered-App_Study_AIxSE25: Replication package for our AIxSE 2025 paper on open-source LLM-powered applications. - github.com," https://github.com/apace-lab/LLM-Powered-App_Study_AIxSE25, [Accessed 15-08-2025].

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

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

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

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

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

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

[27] A. Priyanshu, S. Vijay, A. Kumar, R. Naidu, and F. Mireshghallah, "Are chatbots ready for privacy-sensitive applications? an investigation into input regurgitation and prompt-induced sanitization," *arXiv preprint arXiv:2305.15008*, 2023.

[28] 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.

[29] H. Jin, L. Huang, H. Cai, J. Yan, B. Li, and H. Chen, "From llms to llm-based agents for software engineering: A survey of current, challenges and future," *arXiv preprint arXiv:2408.02479*, 2024.

[30] R. Tufano, A. Mastropaolo, F. Pepe, O. Dabic, M. Di Penta, and G. Bavota, "Unveiling chatgpt's usage in open source projects: A mining-based study," in *Proceedings of the 21st International Conference on Mining Software Repositories*, ser. MSR '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 571–583. [Online]. Available: https://doi.org/10.1145/3643991.3644918

[31] M. Chouchen, N. Bessghaier, M. Begoug, A. Ouni, E. Alomar, and M. W. Mkaouer, "How do software developers use chatgpt? an exploratory study on github pull requests," in *Proceedings of the 21st International Conference on Mining Software Repositories*, 2024, pp. 212–216.

[32] Y. Zhao, X. Hou, S. Wang, and H. Wang, "Llm app store analysis: A vision and roadmap," *arXiv preprint arXiv:2404.12737*, 2024.

[33] Z. Yang, C. Wang, J. Shi, T. Hoang, P. Kochhar, Q. Lu, Z. Xing, and D. Lo, " What Do Users Ask in Open-Source AI Repositories? An Empirical Study of GitHub Issues ," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2023, pp. 79–91. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MSR59073.2023.00024

[34] "OpenAI Developer Community - community.openai.com," https://community.openai.com, [Accessed 07-04-2025].

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

[36] G. Kudrjavets, N. Nagappan, and A. Rastogi, "Do small code changes merge faster? a multi-language empirical investigation," in *Proceedings of the 19th International Conference on Mining Software Repositories*, ser. MSR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 537–548. [Online]. Available: https://doi.org/10.1145/3524842.3528448

[37] K. A. Hasan, M. Macedo, Y. Tian, B. Adams, and S. Ding, "Understanding the time to first response in github pull requests," 2023. [Online]. Available: https://arxiv.org/abs/2304.08426

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

[39] donna.bentley@redis.com, "Building LLM Applications with Kernel Memory and Redis - Redis - redis.io," https://redis.io/blog/building-llm-applications-with-kernel-memory-and-redis/, [Accessed 26-04-2025].

[40] R. S. Sandhu, "Role-based access control," in *Advances in computers*. Elsevier, 1998, vol. 46, pp. 237–286.

[41] K. Zeilenga, "Lightweight directory access protocol (ldap): Technical specification road map," https://www.rfc-editor.org/rfc/rfc4510, Tech. Rep., 2006.

[42] "rag-web-ui/rag-web-ui: RAG Web UI is an intelligent dialogue system based on RAG (Retrieval-Augmented Generation) technology. - github.com," https://github.com/rag-web-ui/rag-web-ui, [Accessed 03-04-2025].

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

[44] "ChuanhuChatGPT/web_assets/javascript/chat-history.js at 8a2fd5681354fdbb2e2e9ee9fbb0aae8d600c453 · GaiZhenbiao/ChuanhuChatGPT - github.com," https://github.com/GaiZhenbiao/ChuanhuChatGPT/blob/main/web_assets/javascript/chat-history.js\#L14, [Accessed 31-05-2024].

[45] "GitHub - lencx/ChatGPT: ChatGPT Desktop Application (Mac, Windows and Linux)," https://github.com/lencx/ChatGPT?tab=readme-ov-file, [Accessed 04-04-2024].

[46] "Microsoft Edge WebView2 — Microsoft Edge Developer - developer.microsoft.com," https://developer.microsoft.com/en-us/microsoft-edge/webview2/?form=MA13LH, [Accessed 16-04-2024].

[47] "GitHub - tw93/Pake: Turn any webpage into a desktop app with Rust." https://github.com/tw93/Pake, [Accessed 04-04-2024].

[48] "sonnylazuardi/chat-ai-desktop: Unofficial ChatGPT desktop app for Mac & Windows menubar using Tauri & Rust," https://github.com/sonnylazuardi/chat-ai-desktop, [Accessed 09-04-2024].

[49] H. Plate, "State of dependency management 2023," https://cdn.prod.website-files.com/6574c9e538a34feac8cec013/65c1a778df2ea35e0edcefc4_State%20of%20Dependency%20Management%202023.pdf, [Accessed 06-11-2024].

[50] dan wesley, "Release notes for Microsoft Edge Security Updates - learn.microsoft.com," https://learn.microsoft.com/en-us/deployedge/microsoft-edge-relnotes-security, [Accessed 07-06-2024].

[51] "xtekky/gpt4free: The official gpt4free repository — various collection of powerful language models - github.com," https://github.com/xtekky/gpt4free, [Accessed 29-05-2024].

[52] "chatpire/chatgpt-web-share: ChatGPT Plus / OpenAI API sharing solution." https://github.com/chatpire/chatgpt-web-share, [Accessed 04-04-2024].

[53] "chatgpt-web-share/frontend/src/router/guard at a9335d53256347352aade0a6ba849818943bbce1 · chatpire/chatgpt-web-share - github.com," https://github.com/chatpire/chatgpt-web-share/tree/a9335d53256347352aade0a6ba849818943bbce1/frontend/src/router/guard, [Accessed 15-07-2025].

[54] "Home — ChatGPT Web Share Docs - cws-docs.pages.dev," https://cws-docs.pages.dev/en/, [Accessed 15-07-2025].

[55] "General Data Protection Regulation (GDPR) – Legal Text - gdpr-info.eu," https://gdpr-info.eu, [Accessed 05-11-2024].

[56] "CodePhiliaX/Chat2DB: AI-driven database tool and SQL client, The hottest GUI client, supporting MySQL, Oracle, PostgreSQL, DB2, SQL Server, DB2, SQLite, H2, ClickHouse, and more. - github.com," https://github.com/CodePhiliaX/Chat2DB, [Accessed 15-07-2025].

[57] "aws-samples/aws-genai-llm-chatbot: A modular and comprehensive solution to deploy a Multi-LLM and Multi-RAG powered chatbot (Amazon Bedrock, Anthropic, HuggingFace, OpenAI, Meta, AI21, Cohere, Mistral) using AWS CDK on AWS - github.com," https://github.com/aws-samples/aws-genai-llm-chatbot, [Accessed 15-07-2025].

[58] "MongoDB: The Developer Data Platform - mongodb.com," https://www.mongodb.com, [Accessed 05-11-2024].

[59] "vulnerability-disclosures/LibreChat/CVE-2024-41703.md at main · realestate-com-au/vulnerability-disclosures - github.com," https://github.com/realestate-com-au/vulnerability-disclosures/blob/main/LibreChat/CVE-2024-41703.md, [Accessed 25-04-2025].

[60] "vulnerability-disclosures/LibreChat/CVE-2024-41704.md at main · realestate-com-au/vulnerability-disclosures - github.com," https://github.com/realestate-com-au/vulnerability-disclosures/blob/main/LibreChat/CVE-2024-41704.md, [Accessed 25-04-2025].