
# LangurTrace: Forensic analysis of local LLM applications

By:
Sungjo Jeong, Sangjin Lee, Jungheum Park

From the proceedings of
The Digital Forensic Research Conference
**DFRWS APAC 2025**
Nov 10-12, 2025

Forensic Science International: Digital Investigation 54 (2025) 301987

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS APAC 2025 - Selected Papers from the 5th Annual Digital Forensics Research Conference APAC

# LangurTrace: Forensic analysis of local LLM applications

Sungjo Jeong, Sangjin Lee, Jungheum Park *

*School of Cybersecurity, Korea University, Seoul, South Korea*

ARTICLE INFO

*Keywords:*
Digital forensics
Large language model (LLM)
Local LLM
Forensic artifact
Forensic tool development

ABSTRACT

A wide variety of applications have been developed to simplify the use of Large Language Models (LLMs), raising the importance of systematically analyzing their forensic artifacts. This study proposes a structured framework for LLM application environments, categorizing applications into backend runtime, client interface, and integrated platform components. Through experimental analysis of representative applications, we identify and classify artifacts such as chat records, uploaded fils, generated files, and model setup histories. These artifacts provide valuable insight into user behavior and intent. For instance, LLM-generated files can serve as direct evidence in criminal investigations, particularly in cases involving the creation or distribution of illicit media, such as CSAM. The structured environment model further enables investigators to anticipate artifacts even in applications not directly analyzed. This study lays a foundational methodology for LLM application forensics, offering practical guidance for forensic investigations. To support practical adoption and reproducibility, we also release `LangurTrace`, an open-source tool that automates the collection and analysis of these artifacts.

## 1. Introduction

Generative artificial intelligence models, particularly Large Language Models (LLMs), have exerted a profound influence across various sectors of society. They are increasingly being adopted in fields such as education, work environments, law, healthcare, counseling, and content creation, which is fundamentally transforming the nature of human–computer interaction. For example, a nationwide survey conducted in late 2023 among 6300 university students in Germany found that almost two-thirds had used AI-based tools in their academic work (Von Garrel and Mayer (2023)). In the mental health domain, Siddals et al. (2024) reported high engagement and positive impacts based on interviews with nineteen individuals who used generative AI chatbots for mental health. Among the most prominent LLM-based services, OpenAI's ChatGPT recently surpassed 400 million weekly active users, according to a February 2025 report by Reuters (2025).

LLMs have primarily been deployed as cloud-based SaaS platforms, such as OpenAI's ChatGPT. However, the consideration of local deployments is enabled by lightweight models and driven by benefits such as privacy and customization (Schillaci (2024)). LLMs typically require high-performance GPUs and large memory capacities. To address this limitation, a line of research has focused on lightweighting techniques for LLMs. Notably, Guo et al. (2025) introduced DeepSeek-R1, an open-source model that significantly reduces GPU and memory requirements while maintaining comparable performance, thereby demonstrating the feasibility of running LLMs on consumer-grade hardware. As platforms such as Hugging Face have made it increasingly easy for users to download and run a wide variety of pre-trained models (PTMs), and as user-friendly local LLM applications continue to be released, accessibility has been further improved.

However, LLMs can also serve as potent tools for criminal activities. For example, cybercriminals are leveraging generative artificial intelligence tools such as ChatGPT to help craft sophisticated and targeted business email compromise (BEC) attacks and other phishing messages (Bob Violino (2023)). In addition, LLMs have been used to generate malicious scripts, such as infostealers and basic forms of ransomware, thereby effectively lowering the technical barrier for cybercriminals (CheckPoint (2023)). Moreover, recent investigations have revealed that generative AI has been exploited to produce synthetic child sexual abuse material (CSAM), raising urgent concerns about this form of exploitation (Sima Kotecha (2025)).

Cases of LLM misuse share similarities with those involving web browsers, suggesting that LLM applications may contain valuable digital artifacts, just as web browsers do. While web browsers significantly improved information accessibility and usability, they have also been exploited for illicit purposes, such as activities on the dark web (Kaur and Randhawa (2020)). Consequently, artifacts such as browsing history, cached content, and search queries from browsers have long been

* Corresponding author.
*E-mail addresses:* sjo98@korea.ac.kr (S. Jeong), sangjin@korea.ac.kr (S. Lee), jungheumpark@korea.ac.kr (J. Park).

https://doi.org/10.1016/j.fsidi.2025.301987

Available online 3 November 2025

regarded as critical sources of evidence in digital forensic investigations (Varol and Sönmez (2017)). Similarly, LLM applications generate distinctive digital traces reflecting a user's intent, decision-making processes, and behavior.

### 1.1. Contributions

Whether delivered through cloud-based SaaS platforms or operated locally, LLM applications consistently generate valuable user interaction artifacts. To the best of our knowledge, this study is the first to present a comprehensive approach for data collection and forensic analysis of both cloud-based and local LLM applications. More specifically, the contributions of this work are as follows:

- We present a structured model of the LLM application environment, categorizing key components and usage patterns to facilitate systematic analysis.
- We identify and categorize key forensic artifacts generated by representative LLM applications, enabling their use in future analysis of similar environments.
- We develop and release `LangurTrace`, an open-source forensic tool that automates the extraction and parsing of LLM application artifacts, supporting reproducibility and operational use.

### 1.2. Outline

This paper is structured as follows: Section 2 reviews related work, covering the intersection of AI and digital forensics and providing technical background on LLM applications. Section 3 characterizes the components of local LLM application environments, including backend runtimes, client interfaces, and integrated platforms. Section 4 presents the results of our forensic analysis, detailing the experimental setup, dataset creation, and artifacts identified across various types of applications. Section 5 introduces the developed forensic tool, describes its design and usage, and demonstrates its effectiveness using the collected dataset. Section 6 discusses the broader implications and limitations of the study, and Section 7 concludes the paper and suggests directions for future research.

## 2. Related work

### 2.1. LLM-assisted digital forensics

Artificial intelligence (AI) has been actively adopted in the field of digital forensics, offering new possibilities for automating evidence analysis, improving investigative accuracy, and reducing human error. Recently, large language models (LLMs) have gained significant attention across various research domains, leading to diverse applications within the field of digital forensics.

Several studies have been conducted on where and how LLMs can contribute to existing digital forensics procedures. For instance, Scanlon et al. (2023) conducted an extensive empirical study on the applicability of ChatGPT (GPT-4) in digital forensic investigations, concluding that while LLMs can assist knowledgeable users, their current limitations necessitate cautious and informed use. Michelet and Breitinger (2024) conducted a case study evaluating how both cloud-based (ChatGPT) and locally deployed (Llama-2) LLMs can assist digital forensic report writing. Xu et al. (2024) presented a hands-on, case-study-driven tutorial that demonstrates how LLMs can be integrated into digital forensic workflows, automating tasks such as suspect profiling based on browser history, summarization of digital artifacts, and knowledge graph reconstruction. This practical approach highlights direct analysis techniques using LLMs for digital forensic purposes. Finally, Wickramasekara et al. (2025a) conducted a comprehensive review on the feasibility of applying LLMs throughout the entire digital forensic investigation process, concluding that LLM adoption can enhance investigative efficiency when applied under appropriate constraints.

Other studies have provided a methodology that can benefit certain areas of digital forensic by utilizing models directly, rather than simply using LLM services. Oh et al. (2024) introduced volGPT, a prompt-engineered LLM framework that combines Volatility plugins and LLM to triage ransomware processes. Wickramasekara and Scanlon (2024) proposed an integrated DF investigation framework built upon Microsoft's AutoGen, leveraging LLMs like LLaMA and StarCoder across multiple collaborative AI agents. These agents decomposed forensic tasks with reduced technical burden on human analysts, showed significant potential to accelerate forensic workflows through agent-based LLM orchestration. Voigt et al. (2024) presented Re-imagen, a framework that leverages LLM to generate coherent user personas and background activities for scenario-based forensic disk images, integrating LLM-driven scripts with VM automation to produce realistic synthetic datasets for digital forensic training and research. Wickramasekara et al. (2025b) introduced AutoDFBench, a benchmarking framework that tests code generated by various LLMs against NIST's CFTT forensic string search standards, highlighting both the potential and limitations of using generative models. Sharma et al. (2025) introduced ForensicLLM, a fine-tuned LLaMA-3.1–8B model, optimized for digital forensics via RAFT (Retrieval Augmented Fine-tuning), showing improved accuracy and attribution over base and RAG (Retrieval-Augmented Generation) models. Kim et al. (2025) proposed SERENA, which LLMs with prompt engineering to extract meaningful insights from semi-structured and unstructured application-to-person (A2P) message data across diverse service providers.

### 2.2. Forensics on LLM-related systems

Large Language Models (LLMs) have been widely adopted as *tools* to assist digital investigations—an area often labeled "LLM for Digital Forensics." In contrast, relatively little research has examined the reverse perspective: conducting digital forensic investigations *on* the AI systems themselves, or "Digital Forensics for AI." The present study follows this latter perspective.

Schneider and Breitinger (2023) proposed an AI-forensics framework that applies grey-box analysis to identify maliciously crafted deep-learning models. Their focus, however, lies in verifying the integrity of the model itself, not in tracing malicious user activities conducted through an LLM.

Other studies treated artifacts left by LLM services as valuable forensic evidence. Dragonas et al. (2024) conducted the first forensic analysis of OpenAI's ChatGPT mobile application, identifying artifacts across Android, iOS, and cloud storage. Cho et al. (2025) carried out a comparative analysis of major cloud-based LLM services such as ChatGPT, Gemini, Copilot, and Claude. These studies highlighted common artifacts generated through user interactions and demonstrated their value in revealing user behavior and intentions. However, their analyses primarily focus on vendor-hosted services, leaving locally deployed LLM scenarios relatively underexamined.

Chernyshev et al. (2023) proposed a digital forensic framework centered on LLM invocation log analysis to detect prompt injection attacks. While their work provided a valuable foundation for detecting such attacks and similarly recognizes locally deployed LLMs and their logs as important forensic artifacts, its scope is primarily limited to threat detection. Complementarily, this study broadens the investigation to encompass a wider variety of artifacts generated by LLM applications, with a particular emphasis on reconstructing user interactions.

## 3. Characterizing the architecture and components of a local LLM system

This section introduces an abstract model of the general LLM application environment and its key components. By examining common usage patterns across various LLM applications, we developed a

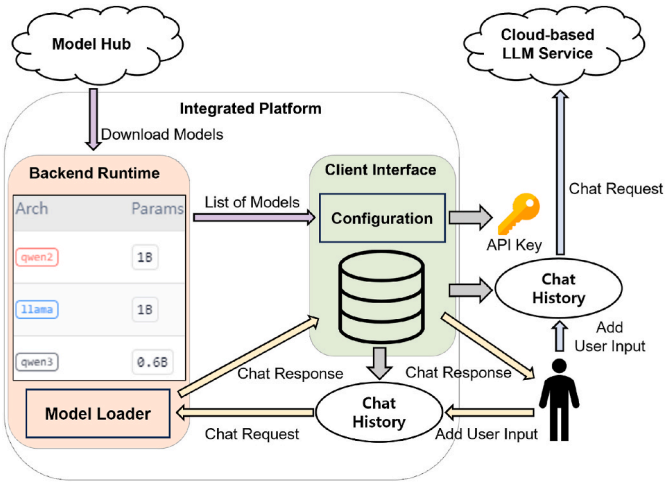


**Fig. 1.** Abstracted architecture and key components of local LLM systems.

generalized model that captures how these systems typically operate. This modeling process enabled the systematic categorization of user interactions and corresponding artifacts observed across different applications.

Fig. 1 provides an overview of typical LLM application environments, encompassing both cloud-based services (e.g., OpenAI's ChatGPT) and local deployments. Users may utilize all or only a subset of these components depending on their needs.

Although the accessibility of local LLMs has improved in recent years, building models from low-level components still requires significant technical expertise. In contrast, cloud-based LLMs are easily accessible via a web browser. Local LLM applications enable users to deploy and use models intuitively, similar to accessing cloud-based services via a web browser.

### 3.1. Backend runtime application and model hub

Model hubs, such as Hugging Face, serve as centralized platforms for hosting, sharing, and distributing pre-trained machine learning models. These repositories facilitate reproducibility and accessibility by allowing users to easily download, fine-tune, or deploy models across a wide range of applications. Recently, the adoption of unified formats such as GGUF (GPT-generated Unified Format) has further improved compatibility and ease of use across tools and platforms. These standardized formats allow consistent loading and execution of language models, particularly in local LLM application environments.

Backend runtime applications enable users to automatically download and deploy models from model hubs with minimal interaction. These systems may also recommend suitable models based on the users' hardware specifications and intended uses. Users can install multiple models concurrently and switch between them as needed. Once a model is deployed, conversations can be initiated directly via a command–line interface (CLI). However, it is common practice to connect the backend to a client interface application through an API, enabling more intuitive interaction. This integration can typically be achieved with just a few clicks.

### 3.2. Client interface application

Client interface applications are graphical user interfaces (GUIs) that enables users to interact with large language models (LLMs). These interfaces commonly resemble cloud-based platforms, such as OpenAI's ChatGPT. Users can create multiple conversation sessions, each configured with a different model and customized system prompt. They can intreract via chat with text or upload files, receiving responses in formats supported by the selected model, such as text or images.

When integrated with local LLMs, these interfaces utilize the APIs exposed by the underlying backend runtime application. Popular backends, such as Ollama, are typically pre-configured with a default API endpoint, including localhost IP addresses and port settings, which allows seamless connection with minimal user intervention. For cloud-based LLMs, integration is typically achieved through the provision of an API key issued by the respective service provider.

### 3.3. Integrated platform application

An integrated platform application refers to software that combines backend runtime functionalities—such as LLM downloading and deployment—and a client interface within a single environment. Although connecting these components via API is relatively straightforward, integrated platforms eliminate even this step, contributing to their increasing popularity. Because both components are integrated into a single application, forensic artifacts from the backend and the client interface appear simultaneously. Further details about these artifacts are provided in Section 4.

### 3.4. Cloud-based LLM

Cloud-based LLMs can be accessed either through a web browser or via local LLM applications using API keys. This dual access method may lead to the misconception that both interfaces share the same conversation context. In practice, even if the API key and browser login belong to the same user account, the conversation contexts are generally managed independently.

Most API-based interactions are stateless—local LLM applications store conversation history locally and supply it to the cloud service with each API call. As a result, the cloud service typically does not persist this context. However, exceptions exist depending on the type of API; certain API implementations may retain limited conversational state or logs associated with API keys for features such as session continuity or audit trails. In contrast, when accessed via a web browser, conversation history is natively preserved within the cloud environment.

Therefore, forensic investigation strategies targeting cloud-based LLMs must consider the access method used. Specifically, cloud forensic analysis is generally not effective when the interaction occurred through a local application using an stateless API key. Further details are discussed in Section 4.

## 4. Forensic artifacts of local LLM environments

Section 4 details the target applications associated with each component of local LLM environments and describes the types of forensic artifacts they generate. Section 4.1 outlines the experimental setup, including the list of analyzed applications. Section 4.2 presents the dataset creation strategy, including representative user scenarios and resulting data. The subsequent sections examine forensic artifacts from three key components of the local LLM stack: backend runtime applications, client interface applications, and integrated platform applications. For each application, we describe the storage locations, data formats, and evidentiary value of the artifacts, with detailed examples provided in dedicated subsections.

### 4.1. Experimental setup

This study aims to identify and analyze user-generated forensic artifacts produced by local LLM applications. To this end, we selected multiple representative applications across different architectural categories, as summarized in Table 1. In the case of ChatGPT, its released version is not clearly defined; instead, it was evaluated during the experimental period from March 1 to April 30, 2025. On the ChatGPT platform, users can obtain an API key through their account and use it

**Table 1**
Target LLM applications.

| Application | Name | Version | Released |
|---|---|---|---|
| Backend Runtime | Ollama | 0.6.5 | 2025-04-06 |
| Client Interface | Chatbox | 1.11.8 | 2025-04-05 |
| Integrated Platform | LM Studio | 0.3.14 | 2025-03-27 |
| Integrated Platform | Msty | 1.8.5 | 2025-03-12 |
| Integrated Platform | Jan | 0.5.16 | 2025-03-14 |
| Integrated Platform | Gpt4All | 3.10.0 | 2025-02-24 |
| Cloud-based LLM | ChatGPT | – | – |

until their purchased token quota is exhausted. We used the GPT-3.5 Turbo and GPT-4o models for text-based interactions, and the standard DALL· E 3 model for image generation. These applications are, to the best of our knowledge, among the most widely adopted in current practice. In the case of integrated platform applications, we selected four representative examples from among many that offer comparable functionality.

To ensure consistency and comparability, all applications were tested on Windows 11 Pro (24H2, build 26100.3775). While our experiments were conducted on Windows, we expect the general types of artifacts to remain consistent across other operating systems such as Linux and macOS. Notably, the specific LLM model used does not significantly affect the types of artifacts generated by each application, as these artifacts are primarily shaped by the application architecture and data handling behavior rather than the model itself.

### 4.2. Dataset creation

User interactions with LLM applications were categorized into specific behavioral patterns, as outlined below. Note that actions described in either the backend runtime applications or the client interface applications also apply to integrated platform applications.

- (Backend)*Local LLM Download and Deletion*: Each downloaded model may either be invoked via the client interface application or remain unused.
- (ClientUI)*Cloud-based LLM Registration and Deletion*: Registering or removing API keys and cloud-based model endpoints. The registered models may or may not be used.
- (ClientUI)*Conversation Session Creation and Deletion*: Refers to the preparation stage for interacting with a model. Session creation involves specifying key parameters such as the selected model and the system prompt.
- (ClientUI)*Conversation*: Sending text(chat) or uploading files (e.g., images, PDFs), and receiving responses in text. Files, such as images, can be generated if supported by the model.

To collect data, we varied the execution order of actions in each run to minimize ordering bias, ensuring that each individual action was performed at least 50 times per application. This design allowed us to observe whether the order or repetition of actions affected the presence, location, or content of the resulting artifacts. Backend runtime and client interface applications were tested in a combined setup, and integrated platforms were used independently. Disk forensics was conducted by monitoring files created, modified, or deleted after each action. Sample data and parsing results are presented in Section 5 using the parsing automation tool.

### 4.3. Types of artifacts in local LLM applications

As a result of our experiments, we identified a variety of artifacts. The most significant ones are summarized in Table 2. Artifacts identified in backend runtime and client interface applications were likewise observed in integrated platform applications. For a detailed breakdown of artifacts by application type is presented in the following subsections.

**Table 2**
Types of common artifacts of LLM applications.

| Artifact Type | Source |
|---|---|
| Downloaded models | Backend runtime |
| Model setup history | Backend runtime |
| Conversation session configurations | Client interface |
| Chat history | Client interface |
| Uploaded/generated files | Client interface |
| API Keys | Client interface |

The detailed file locations of all artifacts are provided in Appendix A.

Model setup history and conversation sessions configurations reveal which models were downloaded and selected. Many pre-trained models are developed for specific purposes. Therefore, identifying which models were selected and when they were downloaded can provide valuable insights into user intent. Conversation content, including chats, uploaded files, and generated files, constitutes the most critical category of artifacts. In particular, generated files may serve as direct evidence in investigations involving the creation or distribution of illicit content, such as CSAM. While API keys related to cloud-based LLM applications may not hold substantial standalone forensic value, they may be shared with service providers as part of a formal cooperation process to request server-side data. Finally, login credentials can be used to directly perform cloud forensic investigations targeting cloud-based LLM applications.

### 4.4. Artifacts of backend runtime applications

LLM models generally do not manage conversations or context on their own, and neither do backend runtime applications. The client interface application retains this data and re-sends it to the backend at each interaction. Accordingly, one of the primary artifacts found in backend runtime applications is the API call logs, which relate to model listing, downloading, deletion, and other operations. Table 3 summarizes the artifacts identified from Ollama.

The most important artifacts are the server log, model manifest, and model layers. The server log contains general backend activity, particularly API call records. API call records include the API type, timestamps, success status, latency, and caller IP (typically 127.0.0.1). Key API operations are model listing, downloading, execution, deletion, and chat requests. Notably, model download logs in Fig. 2 include detailed model metadata, which allows investigators to determine which models were previously installed—even if deleted later. Additional API endpoints are documented in the official Ollama API specification Ollama (2025).

The model manifest is a Docker-style manifest file that describes the model layers, including the model binary, template, license, and parameter files. Each layer is identified by its SHA-256 digest and size. Actual model layers are stored with filenames matching their digest. Investigators can cross-reference these with public model hubs.

Other artifacts are comparatively less critical. The app log captures generic startup and runtime information, but aside from this, it contains little forensic value. The upgrade log documents software update events. The CLI history file stores user-issued requests via the command–line interface in timestamped order, but does not contain model responses.

**Table 3**
Summary of artifacts generated and managed by Ollama

| Artifact | Format | Description |
|---|---|---|
| Server logs | Text | model setup, API calls |
| Model manifest | Docker manifest | Metadata of model layers |
| Model layers | Binary | Template, Parameters |
| App logs | Text | Application usage logs |
| Upgrade logs | Text | Software update history |
| CLI history | Text | User inputs via CLI |

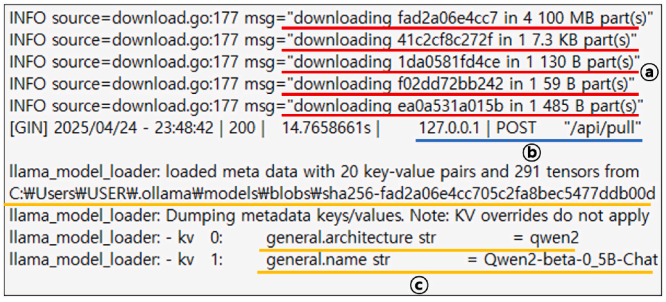


**Fig. 2.** Example of server logs generated by Ollama. (a) Model downloading logs, including the digests and sizes of each corresponding part; (b) API call to `/api/pull` for model download; (c) Model loading (running) logs with file path and model name.

Since most interactions occur via APIs, this artifact holds limited value in practice, except where the user deliberately employs a local CLI-based workflow for specific reasons.

### 4.5. Artifacts of client interface applications

Client interface applications are the primary source of critical forensic artifacts in the LLM application environment. The artifacts include information on the LLM used, system prompt configurations, and conversation contents. Table 4 summarizes the artifacts identified from Chatbox.

The most important artifact is the 'config.json' file. It stores a wide array of data, including configuration details and chat contents. Under the 'settings' key, the value contains configuration information such as API keys, registered local and cloud-based models, and default prompts. The 'chat-sessions' key includes metadata for each session, the system prompt, the selected model, and conversation records. All conversation contents are timestamped in Unix epoch format, and each response is recorded with the model used at the time, as shown in Fig. 3. Uploaded and generated files are noted by filename and MIME (Multipurpose Internet Mail Extensions) type rather than full content. Additionally, Chatbox periodically creates and deletes backup copies of 'config.json', enabling recovery of recent data even after deletion.

Another key set of artifacts is uploaded and generated files, which are stored locally in blob format. Uploaded files are stored as base64-encoded blobs and can be directly decoded. Generated files are embedded as Data URLs within application responses, which are also stored locally, and their content part can be decoded from base64. Notably, even if the user deletes these files through the application interface, the underlying blob data remains in local storage, making recovery straightforward.

Other artifacts include the main log, the API cache, and a model availability store. The main log records the history of config.json backups and software updates. In the API cache, only responses to model listing API calls were found, and not all such requests are guaranteed to be stored. Similarly, the model list—implemented using LevelDB—tracks the list of available models per provider along with their expiration timestamps. Due to the nature of LevelDB, it is sometimes possible to retrieve remnants of previously deleted entries during forensic analysis.

**Table 4**
Summary of artifacts generated and managed by Chatbox

| Artifact | Format | Description |
|---|---|---|
| config.json | JSON | Chat sessions, API keys |
| Uploaded files | Base64-encoded | User-uploaded files |
| Generated files | Base64-encoded | LLM-generated files |
| Main logs | Text | Backup and update history |
| API caches | Chrome cache | Responses to API calls |
| Model list | LevelDB | Model list per provider |

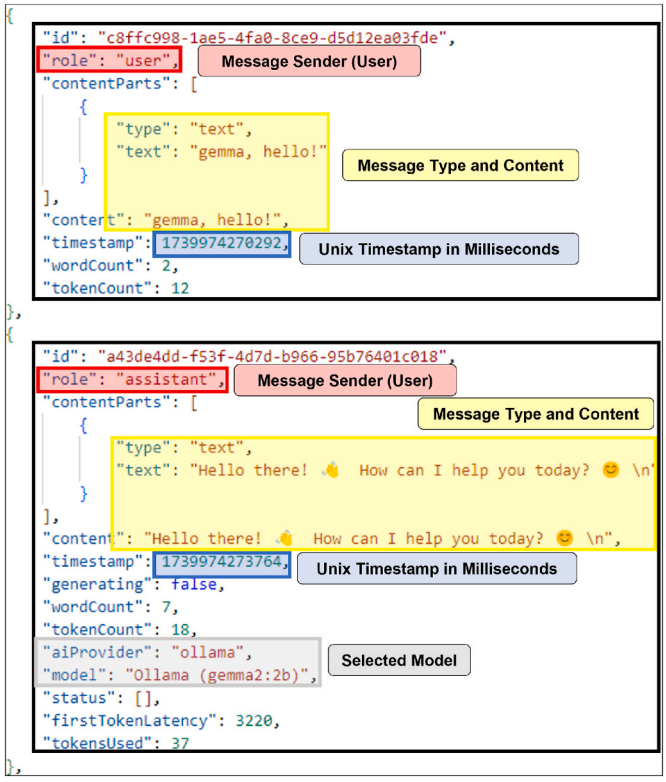


**Fig. 3.** An excerpt from a sample conversation stored in the `chat-sessions` field of `config.json` managed by Chatbox

### 4.6. Integrated platform application artifacts

Integrated platform applications, by definition, contain artifacts from both backend runtime and client interface components, as discussed in Sections 4.4 and 4.5. Since we examined four such applications, we provide a comparative summary of the artifact types in Table 5. It has been confirmed that the key artifacts were found to be identical regardless of provider. For reference, the notations used in Table 5 are as follows:

- ★: Artifacts exist and can be recoverable even after users delete them
- ✩: Artifacts exist but are generally not recoverable once deleted by users
- ×: Not found
- -: Functionality not supported

† "Deletion" refers to actions such as clearing chat history or removing files via the application interface.

The subsequent subsections detail the unique or notable forensic artifacts observed in each application. For a complete list of all identified artifacts across the integrated platform applications introduced in Sections 4.6.1–4.6.4, refer to Appendix B.

**Table 5**
Summary of artifacts from integrated platform applications.

| | LM Studio | Msty | Jan | Gpt4All |
|---|---|---|---|---|
| Downloaded models | ★ | ★ | ★ | ✩ |
| Model setup history | ✩ | ✩ | ✩ | × |
| Chat configurations | ✩ | ★ | ★ | ✩ |
| Chat history | ✩ | ✩ | ★ | ✩ |
| Uploaded files | ✩ | ★ | – | ✩ |
| Generated files | – | – | – | – |
| API Keys | – | ✩ | ★ | ✩ |

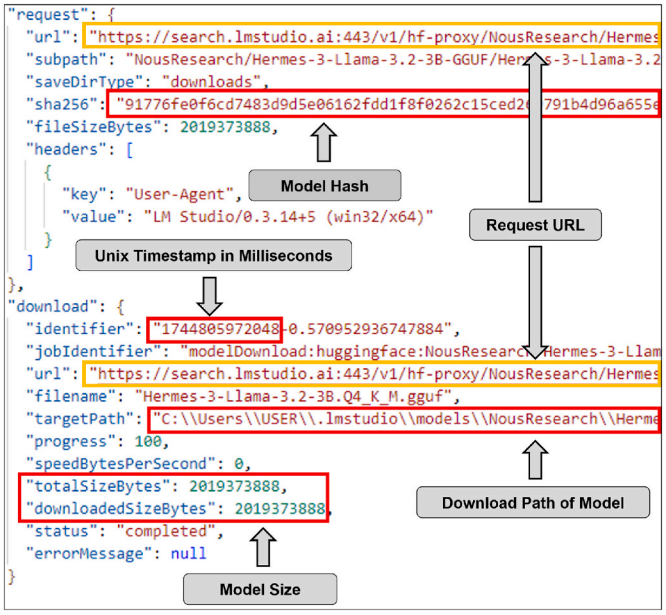


**Fig. 4.** An excerpt from a sample `download-jobs-info.json`

#### 4.6.1. LM studio

LM Studio stored most of its information—such as model setup history, conversations, and metadata of uploaded files—in JSON format, which facilitated artifact extraction. Notably, model setup history was preserved separately in `download-jobs-info.json`, independent of the downloaded model files themselves. As a result, even if a user deleted a model after use, its download history could still be recovered.

`download-jobs-info.json` consists of request and response records related to model downloads, as shown in Fig. 4. The download path of the model can be used to locate the model file on the local machine. If the file has been deleted, the model's name and hash can still serve as identifiers to find the same file elsewhere. In particular, since the request URL is preserved, it is possible to re-access the original web endpoint and retrieve the identical file via the same route used by the user.

#### 4.6.2. Msty

A key characteristic of Msty is that it allows recovery of files uploaded and later deleted by the user. Uploaded files are stored in the attachments storage, and even if the corresponding upload messages are deleted, the files themselves remain intact in storage. In addition, logs related to model setup are recorded in the main log, enabling recovery of deleted model download history, similar to the case of LM Studio.

Notably, a single SQLite 3 database file named `msty.db` contains both the conversation contents and the configuration including API keys. As a result, the majority of artifacts expected from a client interface function can be obtained from this one file, as shown in Table 6.

**Table 6**
Key tables and fields in `msty.db`

| Table | Column | Description |
|---|---|---|
| api_keys | provider | Cloud-based LLM service provider |
| api_keys | key | API key to access cloud-based LLM |
| api_keys | created_at | API key registered time |
| chat_sessions | id | Conversation session ID |
| chat_sessions | title | Conversation session title |
| chat_sessions | created_at | Conversation created time |
| chat_messages | chat_id | Conversation session ID |
| chat_messages | text | Conversation content(message) |
| chat_messages | role | User/AI |
| chat_messages | model_name | Name of used model |
| chat_messages | created_at | Message created time |

#### 4.6.3. Jan

Jan operates on top of the Cortex engine, a local AI engine capable of running across diverse hardware environments, and consequently generates a log file named `cortex.log`. Jan configures the logging level and categories in a rather coarse-grained manner, resulting in highly verbose output. As a result, this log file includes extensive information such as model setup events, conversation content, configuration settings, and even API keys. Therefore, `cortex.log` serves as a key artifact for recovering a wide range of user activities.

For example, in Fig. 5, (a) shows a user's conversation request as recorded in the verbose log. From this, information can be extracted such as the timestamp of the request, its content, and the model used. (b) presents an example log related to the configuration of a cloud-based LLM service, where the user's registered API key is clearly recorded in plaintext. Finally, (c) shows a model download event, including the file path on the local machine and the original URL from which the model was downloaded.

#### 4.6.4. GPT4All

GPT4All records each conversation in its own proprietary format, `.chat`. A notable characteristic is that when a file is uploaded during a conversation, its raw data is fully embedded within the `.chat` file, as illustrated in Fig. 6. Other target applications store only the file path and


**Fig. 5.** An excerpt from a sample verbose log generated and managed by Jan. (a) A conversation request from a user; (b) API key for ChatGPT; (c) Downloading a model from Hugging Face.

metadata in the conversation record while managing the actual file data separately. However, GPT4All embeds the entire file content inline within the `.chat` file, thereby enabling direct recovery of both conversation content (chat) and uploaded files from the binary with minimal pre-processing.

## 5. Implementation and evaluation

### 5.1. LangurTrace: A tool for parsing artifacts generated by local LLM applications in windows

To facilitate the practical application of our findings, we developed an open-source tool named `LangurTrace` (Large language model application user interaction tracer). This tool automates the collection and parsing of artifacts generated by various LLM applications. It is specifically designed to support digital forensic workflows by extracting configuration files, conversation histories, model metadata, and other artifacts identified in Section 4.

`LangurTrace` is designed to integrate seamlessly with a well-known forensic triage tool KAPE (Kroll Artifact Parser and Extractor) (Kroll LLC, 2023). The tool package includes pre-built `Targets`, `Modules`, and a compiled PE (Portable Executable) file (`LangurTrace.exe`), enabling investigators to collect and analyze LLM-related artifacts with minimal configuration. Once deployed, investigators can launch KAPE and select the appropriate targets to retrieve and interpret artifacts from supported applications. While the collection process relies on KAPE, parsing and reporting can be performed independently using the accompanying open-source Python scripts.

Under the hood, `LangurTrace` follows a modular architecture to support its collection and analysis capabilities. The tool is structured into three main modules: the Collector, the Parser, and the Reporter. First, the Collector scans the system for all accessible artifacts related to the target application and copies them to a designated output directory for analysis. Second, the Parser selectively extracts meaningful information from a predefined subset of these artifacts and forwards the results to the Reporter. For example, in the case of Jan, the Parser processes the verbose log file but skips local storage, which contains relatively less meaningful information. Furthermore, within the verbose log, it only extracts chat history and model configuration entries. Finally, the Reporter converts the parsed data into a human-readable report in HTML and other structured formats.

The Collector largely leverages KAPE's built-in functionality, while the Parser and Reporter components are embedded in the provided PE file (`LangurTrace.exe`). The source code, usage instructions, KAPE-compatible components, sample dataset, and output results are publicly available at (Sungjo Jeong, 2025).

Currently, LangurTrace supports a range of LLM applications, including Chatbox, LM Studio, Jan, Msty, Ollama, and GPT4ALL. Additional applications will be supported in future updates.

This tool enhances the reproducibility of our experimental results and provides a practical foundation for automating the forensic analysis of LLM application artifacts.

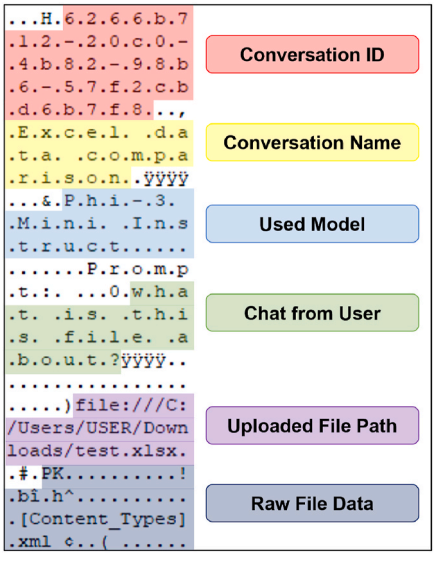


**Fig. 6.** An excerpt from a sample `.chat` file of GPT4All.

### 5.2. Usage and results

The tool can be easily used by copying the provided targets and modules into relevant directories of KAPE. The KAPE offers two main features: `Targets` (Collectors) and `Modules` (Parsers). A Target automatically collects artifacts related to the selected applications from the local system and copies them to a user-defined destination. A Module then parses these collected artifacts, extracting and formatting key information in a way that is easy to review (Kroll LLC, 2023). For reference, detailed setup instructions can be found in the tool's public repository ((Sungjo Jeong, 2025)).

The developed parsing feature mainly focuses on configuration and conversation data. Configuration information is organized in CSV or XLSX format, enabling users to easily identify downloaded or registered models, as well as any stored API keys for cloud-based LLM services. Conversation data is saved as individual HTML files for each session, allowing users to trace the flow of dialogue, as illustrated in Fig. 7. Metadata such as the selected model and the timestamp of each message is appended below each context.

In addition, `LangurTrace` provides application-specific parsing results in structured CSV or XLSX files. For example, in the case of Jan, the `verbose_log_parsed.xlsx` contains separate sheets for chat logs and configuration data, as shown in Fig. 8. These can be used to determine whether a user has deleted specific records. Similarly, for

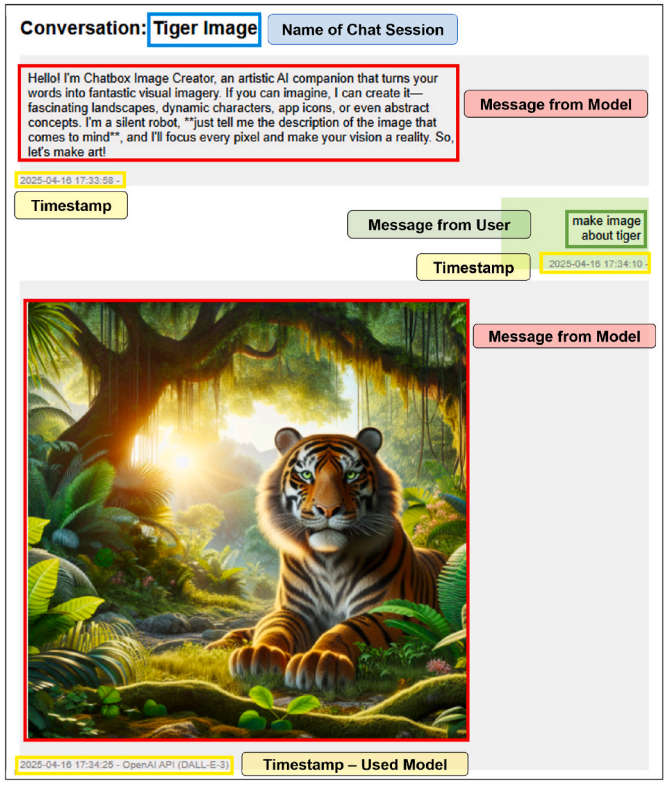


**Fig. 7.** An excerpt from sample conversation records identified and extracted by LangurTrace

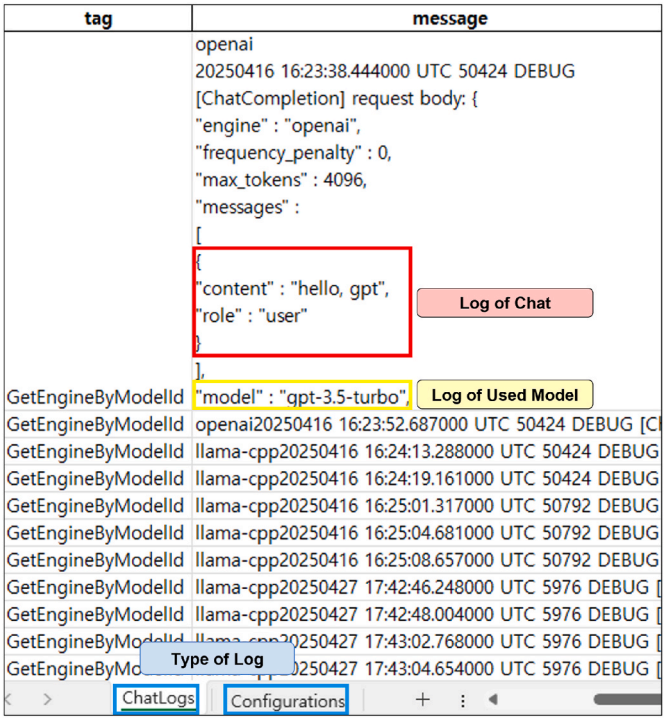


**Fig. 8.** An excerpt from sample verbose logs of Jan identified and extracted by LangurTrace

Msty, the `msty_db.xlsx` includes a `custom_prompts` sheet that reveals the user-defined default prompts. Interpreting these application-specific results may require a deeper understanding of each application's internal structure, beyond what is needed for reviewing the configuration and conversation parsing outputs. If investigators require customized parsing or reporting workflows, the provided Python source code can be used independently of KAPE, enabling flexible adaptation to case-specific needs.

### 5.3. Evaluation

To validate the practical functionality of `LangurTrace`, we tested whether the tool can reliably collect and parse key artifacts from each application, and whether such data remains recoverable even after being deleted through the application's user interface.

Typical user actions—such as downloading models, chatting and uploading files—were performed as summarized in Table 7. In each case, a subset of the generated artifacts was explicitly deleted via the built-in UI. We then examined whether these artifacts remained on disk and whether `LangurTrace` could successfully parse them into structured outputs. Accordingly, Table 8 summarizes the number of artifacts that were successfully recovered and parsed per application, following UI-level deletion by the user. The results for undeleted data are omitted, as all artifacts not deleted through the application's UI were consistently parsed with 100 % success.

More specifically, for Ollama, `server_log.csv` generated by

**Table 7**
Number of user actions for evaluation setup with deleted item counts in parentheses.

| Application | Model Download | Chats | File Uploads | File Generation |
|---|---|---|---|---|
| Backend Runtime | 10 (5) | – | – | – |
| Client Interface | – | 100 (50) | 100 (50) | 100 (50) |
| Integrated Platform | 10 (5) | 100 (50) | 100 (50) | – |

**Table 8**
Recovery results after user-level deletion. Values in parentheses indicate the number of items successfully recovered out of total deleted.

| Application | Deleted Artifacts | | | |
|---|---|---|---|---|
| | Model Download | Chats | File Uploads | File Generation |
| Ollama | 100 %(5/5) | – | – | – |
| Chatbox | – | 58 %(29/50) | 100 %(50/50) | 100 %(50/50) |
| LM Studio | 100 %(5/5) | 0 %(0/50) | 0 %(0/50) | – |
| Msty | 100 %(5/5) | 0 %(0/50) | 100 %(50/50) | – |
| Jan | 100 %(5/5) | 100 %(50/50) | – | – |
| GPT4All | 0 %(0/5) | 0 %(0/50) | – | – |

LangurTrace includes detailed records of model downloads, including timestamps, model names, and manifest information. These allow investigators to identify which models were installed, even after deletion. In Chatbox, although some chat sessions could not be recovered due to the application's non-periodic backup behavior, both uploaded and generated files remained recoverable. These were extracted as Base64-encoded blobs and automatically decoded in the final HTML report. For LM Studio, `model_setup_history.csv` contains model download URLs and timestamps, enabling verification of previously downloaded models. However, since deleted chat records and file uploads left no recoverable traces in the file system, they were also absent from the parsing results generated by `LangurTrace`. For Msty, model setup logs embedded in `main_history.csv` (with the "Model setup" log type) allowed us to identify deleted models. Uploads were also recovered in a separate directory. In Jan, `LangurTrace` extracted and parsed the verbose logs into a structured Excel report. Deleted conversations were listed in the "Chatlogs" sheet, while model download history, including deleted records, appeared in the "Configurations" sheet. Lastly, GPT4All did not yield any recoverable artifacts after deletion. However, undeleted artifacts from all applications including GPT4All were fully parsed and reported, as shown in Figs. 7 and 8.

In summary, the results presented in Table 8 align with the artifacts described in Section 4.4 through Section 4.6, including those listed in Table 5. This consistency confirms that `LangurTrace` successfully identifies most key artifacts that remain recoverable after user-level deletion. However, since the current implementation focuses specifically on key artifacts, manual analysis may still uncover additional information particularly from components previously marked as "generally not recoverable" in earlier sections. For example, API caches in Chatbox are collected by the current version of `LangurTrace`, but are not included in the final report, as they are not classified as key artifacts. Nevertheless, analyzing these caches can, in some cases, reveal model listing API calls, providing potential evidence of which models were downloaded by the user.

## 6. Discussion

### 6.1. Implications of findings

We proposed a structured LLM application environment that contributes to a systematic understanding of how LLMs are used through applications and how data flows across components. Using this approach, we classified various applications into distinct environmental components and investigated the types of forensic artifacts generated by each component. To the best of our knowledge, this is the first study focusing on forensic artifacts generated by different types of local LLM applications.

Artifacts generated by LLM applications provide valuable insights into user behavior and intent, similar to how web browsers have long served as key forensic targets. They include records of conversations

between the user and the LLM, which can be pivotal for understanding user actions. LLM-generated files can serve as direct evidence in criminal investigations, particularly in cases involving the creation or distribution of illicit media, such as CSAM. Additionally, metadata regarding the pre-trained models that users downloaded and utilized can reveal important information about user intent.

Furthermore, applications not explicitly targeted in this study can be mapped to components within the proposed structured environment, enabling the prediction and investigation of potential forensic artifacts based on these results. Although this research primarily focuses on individual users' local environments, the findings are equally applicable to larger-scale organizational users who employ LLMs in similar ways. Differences in available computing resources (e.g., greater GPU capacity) do not fundamentally affect the applicability of these findings.

### 6.2. Limitations

This study primarily focuses on the LLM application environment of individual users. Therefore, the findings may not be directly applicable to non-standard usage scenarios where LLMs are utilized beyond simple conversational interfaces. For instance, organizations or technically skilled users might develop customized front-end services, or multiple users might share a centralized LLM service within an enterprise environment. However, understanding the distinction between artifacts generated by backend runtime and client interface applications can still offer valuable guidance for forensic investigations in such cases.

Due to the wide range of target applications, our experiments were limited to a single operating system (Windows 11 Pro). While the exact format and storage location of artifacts may vary across other operating systems such as Linux or macOS, the types of artifacts and their forensic significance are expected to remain largely consistent.

Our definition of "recoverable" is deliberately confined to disk-level reconstruction achievable with the current `LangurTrace` workflow. We did not evaluate established recovery techniques such as Volume Shadow Copy (VSC) acquisition or live-memory forensics. Consequently, artifacts labeled unable to recover may still be partially retrievable. For structured stores like SQLite 3 or LevelDB, investigators could leverage slack-space analysis or freelist/WAL carving to salvage deleted records. A systematic assessment of these generic data-recovery methods, however, is beyond the scope of this study and is left for future work.

## 7. Conclusion and future directions

This study proposed a structured approach to analyze forensic artifacts from LLM application environments. By categorizing applications into distinct components and experimentally validating artifact types across representative applications, we demonstrated that valuable forensic data can be recovered at both the local and the cloud levels. We also introduced `LangurTrace`, an open-source tool that automates the collection and analysis of these artifacts.

Our findings highlight that chat records, uploaded and generated files, and model usage histories serve as critical evidence of user behavior and intent. Furthermore, the structured environment model enables investigators to anticipate and map artifacts even in applications not directly studied. This work lays foundation for the emerging field of LLM forensics, offering practical methodologies for both individual and organizational investigations.

Future research could extend this study by investigating LLM application artifacts across other operating systems such as Linux and macOS. Additionally, customized LLM usage environments—such as self-developed front-end services or enterprise-level shared LLM platforms—could be explored to broaden applicability. Finally, a more detailed method for leveraging recovered API keys and login credentials in cloud-related forensic investigations could be developed to enhance practical forensic investigations.

## Acknowledgements

This work was supported by the Commercializations Promotion Agency for R&D Outcomes (COMPA) grant funded by the Ministry of Science and ICT (MSIT, Korea) & Korean National Police Agency (KNPA, Korea) (No.RS-2025-02653744).

## Appendix A. Full paths of artifacts related to local LLM applications in Windows

| Application | Artifact | Location |
|---|---|---|
| Ollama | Server logs | %LocalAppData%/Ollama/server.log |
| Ollama | Model manifest | %UserProfile%/.ollama/models/manifests/registry.ollama.ai/library/{model name}/{parameters} |
| Ollama | Model layers | %UserProfile%/.ollama/models/blobs/sha256-{hash digest} |
| Ollama | App logs | %LocalAppData%/Ollama/app.log |
| Ollama | Upgrade logs | %LocalAppData%/Ollama/upgrade.log |
| Ollama | CLI history | %UserProfile%/.ollama/history |
| Chatbox | config.json | %AppData%/xyz.chatboxapp.app/config.json |
| Chatbox | Uploaded files | %AppData%/xyz.chatboxapp.app/chatbox-blobs/{type}input{filename} |
| Chatbox | Generated files | %AppData%/xyz.chatboxapp.app/chatbox-blobs/{type}{filename} |
| Chatbox | Main logs | %AppData%/xyz.chatboxapp.app/logs/main.log |
| Chatbox | API caches | %AppData%/xyz.chatboxapp.app/Cache/Cache_Data |
| LM Studio | Model setup history | %UserProfile%/.lmstudio/.internal/download-jobs-info.json |
| LM Studio | Model files | %UserProfile%/.lmstudio/models/ |
| LM Studio | Conversations | %UserProfile%/.lmstudio/conversations/{ID}.json |
| LM Studio | Uploaded files | %UserProfile%/.lmstudio/user-files/{filename} |
| LM Studio | File metadata | %UserProfile%/.lmstudio/user-files/{filename}.metadata.json |
| LM Studio | Main logs | %AppData%/LM Studio/logs/main.log |
| Msty | Main history | %AppData%/Msty/logs/app.log |
| Msty | Model manifest | %AppData%/Msty/models/manifests/registry.ollama.ai/library/{model name}/{parameters} |
| Msty | Model layers | %AppData%/Msty/models/blobs/sha256-{hash digest} |
| Msty | Conversations | %AppData%/Msty/msty.db |
| Msty | Uploaded files | %AppData%/Msty/attachments/ |
| Jan | Model files | %AppData%/Jan/data/models/{hub name}/{model name}/{parameter}/model.gguf |
| Jan | Model metadata | %AppData%/Jan/data/models/{hub name}/{model name}/{parameter}/model.yml |
| Jan | Model configurations | %AppData%/Jan/data/cortex.db |
| Jan | Chat configurations | %AppData%/Jan/data/threads/{ID}/thread.json |

(*continued on next page*)

(*continued*)

| Application | Artifact | Location |
|---|---|---|
| Jan | Chat sessions | %AppData%/Jan/data/threads/{ID}/messages.jsonl |
| Jan | Verbose logs | %AppData%/Jan/data/logs/cortex.log |
| Jan | Local storage | %AppData%/Jan/Local Storage/leveldb/ |
| GPT4ALL | Model files | %LocalAppData%/nomic.ai/GPT4ALL/{model name}.gguf |
| GPT4ALL | Remote model conf. | %LocalAppData%/nomic.ai/GPT4ALL/gpt4all-{ID}.rmodel |
| GPT4ALL | Conversations | %LocalAppData%/nomic.ai/GPT4ALL/gpt4all-{ID}.chat |

## Appendix B. List of forensically interesting artifacts generated by integrated platform applications

| Application | Artifact | Format | Description |
|---|---|---|---|
| LM Studio | Model setup history | JSON | Model name, Download URL, SHA-256 value, Size |
| LM Studio | Model files | GGUF | GGUF file, Name of model (include deleted) |
| LM Studio | Conversations | JSON | Configurations, Chat sessions |
| LM Studio | Uploaded files | Original format | Uploaded files |
| LM Studio | File metadata | JSON | Type, Size, Original name, SHA-256 value |
| LM Studio | Main logs | Text | Update log, Error log |
| Msty | Main history | Text | Model setup, Model loading history, Attachment upload and deletion history |
| Msty | Model manifest | Docker V2 manifest | Metadata of model layers |
| Msty | Model layers | Binary | Model, Template, License, Parameters |
| Msty | Conversations | SQLite 3 | Configurations, Chat sessions, API keys |
| Msty | Uploaded files | Original format | Uploaded files (include deleted) |
| Jan | Model files | GGUF | GGUF file |
| Jan | Model metadata | YAML | Metadata of model |
| Jan | Model configurations | SQLite 3 | API Keys, Model information |
| Jan | Chat configurations | JSON | Chat session name, Selected model |
| Jan | Chat sessions | JSONL | Chat history, Selected model, Timestamps |
| Jan | Verbose logs | Text | Model setup history, Chat sessions, API keys(include deleted) |
| Jan | Local storage | LevelDB | Chat sessions, Downloaded models (include part of deleted) |
| GPT4all | Model files | GGUF | GGUF file |
| GPT4all | Remote model conf. | JSON | API Key, Model provider |
| GPT4all | Conversations | .chat (proprietary) | Configurations, Chat sessions, Uploaded file |

## References


Bob Violino, 2023. AI Tools Such as ChatGPT Are Generating a Mammoth Increase in Malicious Phishing Emails. URL: cnb.cx/46zoo0z. (Accessed 20 April 2025).

CheckPoint, 2023. OpwnAI: cybercriminals starting to use ChatGPT. URL: https://research.checkpoint.com/2023/opwnai-cybercriminals-starting-to-use-chatgpt/. (Accessed 20 April 2025).

Chernyshev, M., Baig, Z., Doss, R.R.M., 2023. Towards Large Language model (LLM) forensics using LLM-based invocation log analysis. In: Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis, pp. 89–96.

Cho, K., Park, Y., Kim, J., Kim, B., Jeong, D., 2025. Conversational AI forensics: a case study on ChatGPT, Gemini, Copilot, and claude. Forensic Sci. Int.: Digit. Invest. 52, 301855.

Dragonas, E., Lambrinoudakis, C., Nakoutis, P., 2024. Forensic analysis of OpenAI's ChatGPT mobile application. Forensic Sci. Int.: Digit. Invest. 50, 301801.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al., 2025. Deepseek-r1: incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948.

Kaur, S., Randhawa, S., 2020. Dark web: a web of crimes. Wirel. Pers. Commun. 112, 2131–2158.

Kim, J., Jeong, B., Park, S., Lee, S., Park, J., 2025. Your forensic AI-assistant, SERENA: systematic extraction and reconstruction for enhanced A2P message forensics. Forensic Sci. Int.: Digit. Invest. 53, 301931.

Kroll LLC, 2023. KAPE: kroll artifact parser and extractor. https://www.kroll.com/en/insights/publications/cyber/kroll-artifact-parser-extractor-kape. (Accessed 5 May 2025).

Michelet, G., Breitinger, F., 2024. ChatGPT, Llama, can you write my report? An experiment on assisted digital forensics reports written using (local) large language models. Forensic Sci. Int.: Digit. Invest. 48, 301683.

Oh, D.B., Kim, D., Kim, H.K., 2024. volGPT: evaluation on triaging ransomware process in memory forensics with Large Language Model. Forensic Sci. Int.: Digit. Invest. 49, 301756.

Ollama, 2025. Ollama API documentation. https://github.com/ollama/ollama/blob/main/docs/api.md. (Accessed 24 April 2025).

Reuters, 2025. OpenAI's weekly active users surpass 400 million. URL: https://www.reuters.com/technology/artificial-intelligence/openais-weekly-active-users-surpass-400-million-2025-02-20/. . (Accessed 20 April 2025).

Scanlon, M., Breitinger, F., Hargreaves, C., Hilgert, J.N., Sheppard, J., 2023. ChatGPT for digital forensic investigation: the good, the bad, and the unknown. Forensic Sci. Int.: Digit. Invest. 46, 301609.

Schillaci, Z., 2024. LLM adoption trends and associated risks. In: Large Language Models in Cybersecurity: Threats, Exposure and Mitigation. Springer Nature, Switzerland Cham, pp. 121–128.

Schneider, J., Breitinger, F., 2023. Towards AI forensics: did the artificial intelligence system do it? J. Inf. Secur. Appl. 76, 103517.

Sharma, B., Ghawaly, J., McCleary, K., Webb, A.M., Baggili, I., 2025. ForensicLLM: a local large language model for digital forensics. Forensic Sci. Int.: Digit. Invest. 52, 301872.

Siddals, S., Torous, J., Coxon, A., 2024. "It happened to be the perfect thing": experiences of generative AI chatbots for mental health. npj Mental Health Research 3, 48.

Sima Kotecha, 2025. AI-generated child sex abuse images targeted with new laws. URL: https://www.bbc.com/news/articles/c8d90qe4nylo. (Accessed 20 April 2025).

Sungjo Jeong, 2025. LangurTrace. https://github.com/jeongramon/LangurTrace. (Accessed 21 July 2025).

Varol, A., Sönmez, Y.Ü., 2017. The importance of web activities for computer forensics. In: 2017 International Conference on Computer Science and Engineering (UBMK). IEEE, pp. 66–71.

Voigt, L.L., Freiling, F., Hargreaves, C.J., 2024. Re-imagen: generating coherent background activity in synthetic scenario-based forensic datasets using large language models. Forensic Sci. Int.: Digit. Invest. 50, 301805.

Von Garrel, J., Mayer, J., 2023. Artificial Intelligence in studies—use of ChatGPT and AI-based tools among students in Germany. Humanities and social sciences communications 10, 1–9.

Wickramasekara, A., Breitinger, F., Scanlon, M., 2025a. Exploring the potential of large language models for improving digital forensic investigation efficiency. Forensic Sci. Int.: Digit. Invest. 52, 301859.

Wickramasekara, A., Densmore, A., Breitinger, F., Studiawan, H., Scanlon, M., 2025b. AutoDFBench: a framework for AI generated digital forensic code and tool testing and evaluation. In: Proceedings of the Digital Forensics Doctoral Symposium, pp. 1–7.

Wickramasekara, A., Scanlon, M., 2024. A framework for integrated digital forensic investigation employing AutoGen AI agents. In: 2024 12th International Symposium on Digital Forensics and Security (ISDFS). IEEE, pp. 1–6.

Xu, E., Zhang, W., Xu, W., 2024. Transforming digital forensics with Large Language Models: unlocking automation, insights, and justice. In: Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, pp. 5543–5546.