AudAgent: Automated Auditing of Privacy Policy Compliance in AI Agents

Ye Zheng
Rochester Institute of Technology
Rochester, NY, USA

Yimin Chen
University of Massachusetts Lowell
Lowell, MA, USA

Yidan Hu
Rochester Institute of Technology
Rochester, NY, USA

Abstract

AI agents can autonomously perform tasks and, often without explicit user consent, collect or disclose users' sensitive local data, which raises serious privacy concerns. Although AI agents' privacy policies describe their intended data practices, there remains limited transparency and accountability about whether runtime behavior matches those policies. To bridge this gap, we present AudAgent, a tool that continuously monitors AI agents' data practices in real time and guards compliance with their stated privacy policies.

AudAgent comprises four components for automated privacy auditing of AI agents. (i) Policy formalization: a novel cross-LLM voting mechanism that ensures high-confidence parsing of privacy policies into formal models. (ii) Runtime annotation: a lightweight Presidio-based analyzer that detects sensitive data and annotates data practices based on the AI agent's context and the formalized privacy policy model. (iii) Compliance auditing: ontology graphs and automata-based checking that link the privacy policy model with runtime annotations, enabling on-the-fly compliance verification. (iv) User interface: an infrastructure-independent implementation that visualizes the real-time execution trace of AI agents alongside detected privacy violations, providing user-friendly transparency and accountability.

We evaluate AudAgent on AI agents built with mainstream frameworks, demonstrating its effectiveness in detecting and visualizing privacy policy violations. Using AudAgent, we further find that many privacy policies lack explicit safeguards for highly sensitive data such as SSNs, whose misuse violates legal requirements, and that many agents—including those powered by Claude, Gemini, and DeepSeek—do not refuse to process such data via third-party tools. AudAgent proactively blocks operations on such data, overriding the agents' original privacy policies and behavior. These results highlight that AudAgent complements existing AI agents with local privacy protection, enabling trustworthy deployments.

Keywords

privacy policy compliance, AI agents, runtime monitoring

1 Introduction

AI agents [5, 7, 8], including intelligent assistants and workflow-automation tools powered by large language models (LLMs), are increasingly able to autonomously perform a broadening range of personal and professional tasks on users' behalf. To operate, these agents often collect, process, and sometimes disclose users' local data [19–21, 24], which may contain sensitive information [32, 45, 53], often without explicit user consent. Although such data practices are described in the platforms' privacy policies, users often lack clear visibility into whether an agent's runtime behavior actually conforms to those privacy policies. This opacity is further compounded by complex third-party integrations, which may involve external APIs and services that can lead to unintended data collection and disclosure. Such challenges highlight the need for effective auditing tools that empower users to verify and ensure that their AI agents adhere to the stated privacy policies or users' privacy preferences.

Related work. From a broader perspective, defending security and privacy in AI agents against various threats, such as extracting sensitive data via malicious prompts [46, 50, 52, 68] and compromised tools [39, 72, 85], has led to static analysis of agent components [15, 17], sandbox-based behavioral testing [74, 91]. These works on defending privacy in AI agents largely focused on benchmarking vulnerabilities and malicious behaviors, while this paper focuses on end users' privacy. Technically, this paper also relates to privacy policy analysis [32, 34, 86, 93] and compliance auditing of applications [30, 31, 94]. Among them, existing compliance-auditing research concentrated on mobile applications, web services, and similar domains, while this paper targets AI agents. Appendix A provides detailed discussion and comparison with related works. In summary, no effective real-time method currently exists that enables end users to determine whether and how their AI agents handle sensitive data and whether those practices comply with stated privacy policies. This gap motivates our research question: *How can end users audit their AI agents' runtime data practices against the privacy policies agents claim to follow or users expect?*

Challenges. Enabling end-user auditing of AI agents' data practices is non-trivial because of the gap between high-level, natural-language privacy policies and low-level, unordered runtime data practices. Specifically, there are four main challenges:

- *(Policy formalization)* Privacy policies are typically written in natural language and must be translated into precise, machine-checkable formats.
- *(Limited visibility)* AI agents frequently interact with local and third-party services, such as APIs and file systems, making it difficult to detect sensitive data practices from a single vantage point.
- *(Effort and efficiency)* Auditing should be automated to avoid expert involvement and should be real-time without significant overhead.
- *(User-friendly visualization)* The tool should offer an intuitive, infrastructure-agnostic visualization, making privacy auditing accessible and facilitating broad adoption.

Proceedings on Privacy Enhancing Technologies 2026(3), 177–199

https://doi.org/10.56553/popets-2026-0077

This paper. We propose AudAgent, an automated data-auditing tool that tackles these challenges. (i) AudAgent leverages the regular structure of privacy policy documents to guide LLMs in extracting a formal, machine-checkable privacy policy model that captures data categories and constraints on collection, processing, disclosure, and retention. To improve extraction robustness and reduce errors, AudAgent applies cross-LLM voting for confidence aggregation to reconcile the policy elements. (ii) AudAgent employs a lightweight, Presidio-based [57] local analyzer to monitor the agent's inbound and outbound data in real time, combining runtime context with the extracted policy model to identify and annotate sensitive data usage. (iii) AudAgent uses ontology graphs and automata-based checking to connect the privacy policy model with runtime annotations: the ontology graphs reconcile hierarchical inclusions between the terms in the privacy policy model and runtime data, and the automata-based checking compiles the privacy policy model into intuitive state machines for efficient, on-the-fly compliance auditing. (iv) AudAgent is implemented as a modular, OS-agnostic plug-in with a browser-based interface that visualizes agents' real-time execution traces and highlights privacy violations detected during auditing, showing an intuitive view of data flows, transparency, and accountability.

AudAgent is fully automated: it only requires relevant privacy policy documents. When run alongside an AI agent, it continuously monitors agent behavior in real time and alerts users to potential privacy violations. Figure 1 illustrates the overall workflow.

Deployment examples. We implement AudAgent as a Python module that can be integrated with mainstream AI agent building frameworks (orchestration platforms). Section 5 presents case studies using Microsoft's AutoGen [55], an open-source framework for building LLM-based agents with tool support. AudAgent also supports popular framework LangChain [11] and Model Context Protocol (MCP) [12]. Our deployments show that AudAgent complements these frameworks and protocols by adding real-time privacy auditing and visualization, addressing their lack of built-in auditing mechanisms.

Benefits to users and platforms. For users, AudAgent provides real-time privacy auditing for their AI agents, along with configurable controls over specific data-leakage behaviors (e.g. keyword filters or category-based rules in user-defined policies). This customization and transparency help users detect risky data flows and enforce privacy preferences that were previously difficult to realize. For AI-agent platforms, AudAgent serves as a diagnostic and accountability tool: it helps assess whether deployed agents' runtime behavior aligns with stated privacy policies, surface discrepancies, and inform updates to data-handling practices or policy language to strengthen compliance and user trust. By proactively identifying privacy and regulatory risks, this design supports more responsible AI deployment.

AudAgent's contributions can be summarized as the following three aspects:

- To our knowledge, AudAgent is the first tool that enables automated auditing of AI agents' data practices against privacy policy documents. It provides end users with transparency to verify that their agents' behavior matches stated privacy expectations.
- AudAgent addresses challenges in policy formalization, limited visibility, efficient auditing, and user-friendly visualization. It bridges the technical gap between natural-language privacy policies and runtime data practices of AI agents, facilitating deployments of AI agents by enhancing user trust and accountability.
- We implement AudAgent as a plug-in module that can agnostically run with various AI agent building frameworks. Experiments demonstrate that AudAgent effectively identifies potential privacy policy violations in real time. Using AudAgent, we further find that many privacy policies lack explicit safeguards for highly sensitive data such as SSNs, whose misuse violates legal requirements, and that many agents—including those powered by Claude, Gemini, and DeepSeek—do not refuse to process such data via third-party tools. AudAgent proactively blocks operations on such data, overriding the agents' original privacy policies and behavior.

Structure. The main parts of this paper are organized as follows. Section 2 formalizes the problem and provides preliminaries. Section 3 presents the design of AudAgent, detailing its architecture and core components. Following that, Section 4 discusses AudAgent's technical details, then Section 5 presents experiments, demonstrating AudAgent's impact on existing AI agents, along with ablation studies evaluating its key components.

2 Preliminaries

This section specifies the threat model, and reviews privacy policies, AI agents, privacy policy compliance, and real-time visualization.

2.1 Assumptions and Threat Model

Given the varying definitions of AI agents and the practical limits of observing data flows, we make the following scoping assumptions:

- An AI agent consists of an LLM and a set of tools: the LLM provides core decision-making, while tools interact with context or external systems to extend the agent's capabilities, i.e. "LLM agents" in some literature [89].
- The agent's control logic (orchestrator) executes locally on the user's device, whereas the LLM and third-party tools may be hosted as cloud services.

These assumptions reflect common agent architectures [11, 18, 55, 89], where a local orchestrator coordinates on-device and cloud-based components.

Threat model. We assume the LLM provider and third-party tool services are honest-but-curious. They follow the protocols specified by the agent's orchestrator, i.e. only have access to data sent to them and return responses. However, when performing tasks, the AI agent controlled by the LLM may misuse or disclose sensitive user data beyond the limits stated in the agent's privacy policy. For example, an agent might forward sensitive information to unwarranted third-party tools or otherwise behave in ways that deviate from user expectations or its declared data practices.* Our

*This risk is common when using existing AI-agent frameworks (e.g. AutoGen, LangChain) and protocols (e.g. MCP). Typically, although MCP decouples tool calls, which may contain sensitive data, from LLMs and thereby reduces exposure to LLM providers, agents may still send sensitive data to unauthorized third-party tools.

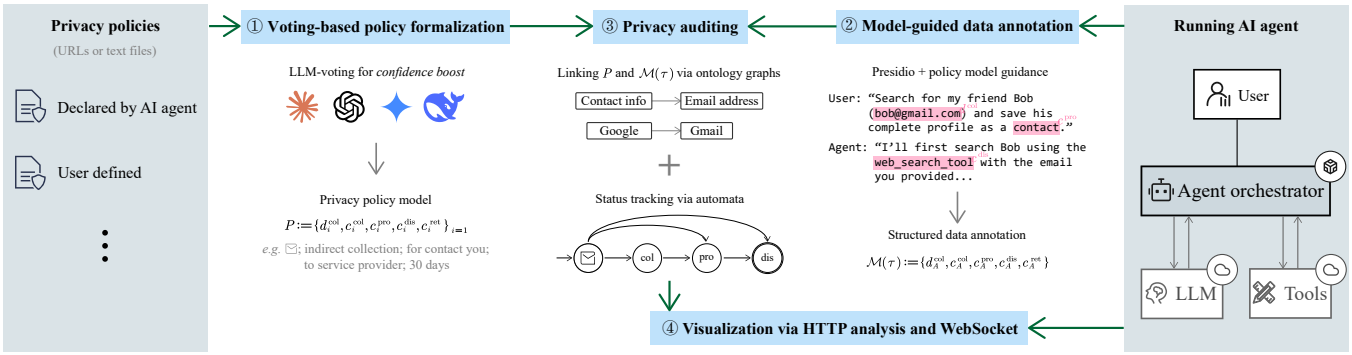

Figure 1: Overview of AudAgent, a tool for auditing an AI agent's data practices against specified privacy policies. It comprises four components: (1) Voting-based policy formalization (Section 3.1), which performs a one-time extraction of privacy policy documents into a formal model; (2) Model-guided data annotation (Section 3.2), which continuously monitors and annotates the agent's data practices; (3) Privacy auditing (Section 3.3), which performs on-the-fly checks comparing the policy model with runtime annotations; and (4) Visualization via HTTP analysis and WebSocket (Section 3.4), which visualizes the agent's real-time execution trace and highlights potential violations detected during auditing.[*]

goal is to enable end users to detect privacy risks and potential policy violations arising from the agent's data flows in real time.

To achieve this goal, AudAgent must address challenges from four aspects: privacy policy formalization, AI agents' data interactions, compliance auditing, and real-time visualization. The following subsections examine these aspects in detail and highlight the associated challenge.

2.2 Privacy Policies

According to the General Data Protection Regulation (GDPR) [4], *any* organization that processes personal data of residents must provide clear information about how that data is used.[†] Privacy policies are legal documents for this purpose, that explain how an organization collects, uses, and discloses personal information from users. Privacy policies typically follow a written structure of sections about data collection, processing, disclosure, and retention [40, 84?]. Formally, we model a privacy policy as follows.

Definition 1 (Privacy policy). *A privacy policy P is a data model defined by a set of tuples $P := \{(d_i^{\text{col}}, c_i^{\text{col}}, c_i^{\text{pro}}, c_i^{\text{dis}}, c_i^{\text{ret}})\}_{i=1}$,[‡] where each i corresponds to a specific data type (category) and its associated practices:*

- $d_i^{\text{col}} \in D^{\text{col}}$ *is a data type the first party collected from users together with the conditions c_i^{col} under which collection occurs;*
- $c_i^{\text{pro}} \in C^{\text{pro}}$ *specifies the purposes or conditions the first party processes the data d_i^{col};*
- $c_i^{\text{dis}} \in C^{\text{dis}}$ *specifies the third parties to whom the first party discloses the data;*
- $c_i^{\text{ret}} \in C^{\text{ret}}$ *denotes retention (expiration) periods associated with collected data type d_i^{col}.*

This five-element tuple captures the structure of GDPR-style privacy policies [4, 69] as well as templates produced by common privacy-policy generation tools [41, 81]. For example, the first four sections of OpenAI's privacy policy [65] map directly to the components defined above.[§] Concretely, the policy states that it collects users' location information (d^{col}) such as IP address or device GPS from using their services (c^{col}), processes this data to provide users more accurate responses (c^{pro}), and may disclose it to hosting providers and customer-support vendors (c^{dis}). The policy also states that, chats data will be retained for up to 30 days (c^{ret}). In addition to this retention policy, for users who opt in to the Zero Data Retention API [64], the service does not retain their data.

Challenge 1: Privacy formalization. Although the OpenAI example demonstrates a direct mapping between a privacy policy and our formal model, real-world privacy policies are written in natural language with widely varying formats and terminology. They typically lack the concise and normalized structure required for machine interpretation, making it difficult to automatically extract the components for our formal privacy policy model.

2.3 Data Interactions of AI Agents

To fulfill the tasks prompted by users, AI agents often perform multiple rounds of interaction with the user and external tools. In this paper we focus on the data interactions; an agent's data interactions during a task execution can be abstracted below.

Definition 2 (Data interactions of an AI agent). *An AI agent's data interactions during a task execution are modeled by a finite sequence $\langle d_1, d_2, \ldots, d_N \rangle$, where each d_i is a data item and can be classified into one of four categories:*

- user→agent*: data provided by or collected from the user (e.g. prompts, personal files);*
- user←agent*: data processed and produced by the agent and returned to the user (e.g. generated responses);*

[*] See Figure 5 for a screenshot and https://github.com/ZhengYeah/AudAgent for a demonstration video of AudAgent. Table 1 compares AudAgent with existing AI-agent visualization tools.

[†] Other well-known regulations, such as the California Consumer Privacy Act (CCPA) [?], have similar requirements. Meanwhile, most organizations publish privacy policies to build trust with their users.

[‡] (i) Some privacy policies include additional components (e.g. organization identifiers, user rights, or children's data handling). We focus on the core data-practice components for clarity. (ii) We use such five-element model for auditing; the value domains of these elements in our auditing model are specified in Section 3.2.2.

[§] Although OpenAI's privacy policy describes data types and their associated conditions across separate sections, they can easily be paired by matching related terminology. Section 4 further details how OpenAI's policy maps to the five-element tuple in Definition 1 and discusses the expressiveness of Definition 1.

- agent→third:[¶] *data sent to external services or tools (e.g. tool calls, API requests);*
- agent←third*: data received from external services or tools (e.g. tool outputs).*

Among these data interactions, the data flow from the user to the agent (user→agent) and from the agent to third parties (agent→third) are of particular interest, as they represent the points where users' sensitive information may be outside users' control and potentially violated the agent's privacy policy. The other two categories (user←agent and agent←third) are also relevant, as they may imply data processing and retention practices of the agent.

Challenge 2: Limited visibility. Although this categorization clarifies the structure of agent data flows in principle, identifying these flows from an end user's perspective remains challenging. In practice, AI agents interact through largely unstructured prompts and responses, which complicates accurate and efficient identification of sensitive data practices. Moreover, agent executions often span multiple rounds of communication and tool use, making it difficult to track data flows over time and detect potential disclosures as they occur.

2.4 Privacy Policy Compliance

Although an AI agent may provide a privacy policy P that outlines its data collection[‖] and disclosure practices, users still lack transparency into what data is actually collected and disclosed during execution. This gap raises concerns about whether the agent complies with its stated privacy policy and aligns with users' privacy expectations.

Definition 3 (Privacy policy compliance for an AI agent). *Consider a single task execution of an AI agent A and an observed data-practice tuple $(d_A^{\text{col}}, c_A^{\text{col}}, c_A^{\text{pro}}, c_A^{\text{dis}}, c_A^{\text{ret}})$, where d_A^{col} is the collected data item and c_A^{col}, c_A^{pro}, c_A^{dis}, and c_A^{ret} denote the corresponding collection conditions, processing purposes/conditions, disclosure recipients, and retention period, respectively. We say that A is* compliant *with a privacy policy P if*

$$P \models \left(d_A^{\text{col}}, c_A^{\text{col}}, c_A^{\text{pro}}, c_A^{\text{dis}}, c_A^{\text{ret}}\right),$$

i.e. P semantically permits (entails) the specified data practice.

Auditing the entailment relation in Definition 3 is difficult in practice, even assuming the privacy policy model P and the collected data types d_A^{col} are identified accurately. The difficulty stems from the volume of data items produced during execution, their syntactic variability, and temporal dependencies across interaction steps.

Challenge 3: Effort and efficiency. Although Definition 3 formally connects the privacy policy model with annotated execution data, a practical gap remains in the effort and efficiency required to perform auditing. Bridging this gap requires an automated auditing that is accurate, low-overhead, and requires minimal user intervention. Moreover, task executions may generate a large number of data items, necessitating low-latency mechanisms suitable for real-time auditing.

2.5 Real-time Visualization

User trust in AI agents depends not only on their outputs (as with chatbots), but also on transparency into the agent's internal processes and data practices [33, 71]. Real-time visualization provides such transparency by mapping observed runtime events (e.g. external requests) to intuitive visual representations (e.g. graphs). This enables users to (i) see what data is accessed or disclosed at each step and (ii) detect mismatches where runtime behavior appears to exceed expectations. Accordingly, post hoc logs are insufficient: users need to identify privacy risks in real time and intervene when necessary.

Real-time visualization architectures typically comprise three layers: a data-collection layer that captures the agent's interactions with users, context, and external services as they occur; a streaming layer that filters, normalizes, and routes captured events as they occur; and a visualization layer that renders these events to users in real time. Beyond the standard engineering concerns, achieving a user-friendly visualization for AI agents requires addressing additional challenges.

Challenge 4: User-friendly visualization. To be broadly usable, real-time visualization should remain independent of specific agent frameworks and operating systems. Framework independence avoids the visualization being tied to a specific agent architecture, ensuring compatibility across diverse agent implementations. Operating-system independence enables deployment across platforms and reduces setup friction for end users.

In summary, these four challenges highlight the need for a tool that supports clear privacy formalization, real-time visibility, efficient auditing, and user-friendly visualization. AudAgent is designed to meet these needs and enable users to understand and audit their AI agents' data practices.

3 AudAgent

This section describes the design of AudAgent. To address the four challenges identified above, AudAgent incorporates the following design elements:

- *(Voting-based policy formalization)* AudAgent employs multiple LLMs to independently interpret natural-language privacy policy documents, and then aggregates their outputs via semantic equivalence checking and majority voting. This one-time voting step yields a final privacy policy model with improved accuracy and a quantifiable confidence boost.
- *(Model-guided data annotation)* AudAgent performs data annotation methods guided by the privacy policy model in Definition 1, tailored separately for the collection, processing, disclosure, and retention stages. These methods complement the lightweight and local Presidio [57] analyzer for identifying sensitive data types, enabling low-overhead annotation during AI agent execution.
- *(Privacy auditing via ontology graph and automata)* AudAgent leverages ontology alignment to bridge hierarchical granularity mismatches between terms in the privacy policy model and runtime observations, and compiles the privacy

[¶] Here, the "first party" refers to (the LLM component of) the AI agent that directly interacts with the user; "third parties" are the external services or tools defined in the agent's privacy policy, which do not directly interact with the user but may receive data from the first party.

[‖] If a user includes sensitive data in a prompt, we treat that data as collected. Such passive collection is often explicitly stated in AI-agent privacy policies, e.g. Anthropic's Claude [25] and AutoGPT [26].

policy model into intuitive state machines that enable on-the-fly compliance auditing.
- *(Visualization via HTTP analysis and WebSocket)* AudAgent integrates a real-time data collection layer using HTTP analysis to capture the AI agent's data interactions, a streaming layer using WebSocket to transmit annotated data and audit results, and a visualization layer based on web-browser frontends, which together ensure independence of agent frameworks and operating systems.

Figure 1 illustrates the overall workflow of AudAgent, consisting of the above four components.

3.1 Voting-Based Policy Formalization

Privacy policy formalization is the first step in AudAgent, which transforms a natural-language privacy policy document into a structured, machine-auditable model as defined in Definition 1. Manual formalization (or analysis) requires expert knowledge and is labor-intensive, while statistical and symbolic extraction methods [22, 82] may struggle with the complexity and variability of natural language. LLM-based approaches have achieved state-of-the-art performance in privacy policy analysis [32, 82, 86, 88], owing to their superior natural-language understanding capabilities. Nonetheless, these methods typically rely on a single LLM, which can introduce inaccuracies due to model-specific biases or occasional errors, and they provide no quantifiable *confidence* measurement for the correctness of the output. To enhance the accuracy and confidence of the result, we design a majority voting approach that leverages multiple LLMs, ensuring a comprehensive understanding of the document. In this process, one challenge is to identify semantically equivalent elements across different LLM outputs.

3.1.1 LLM Voting. AudAgent's LLM voting approach for privacy policy formalization simulates a multi-participant decision-making process, with each LLM contributing its own interpretation of the policy. The final policy model is derived from the consensus among the LLMs, enhancing both accuracy and confidence in the result.

Specifically, given a privacy policy document and M different LLMs $\{L_1, L_2, \ldots, L_M\}$, each LLM is prompted independently to analyze the document,** and produce its interpretation P_m following the template in Definition 1. Elements in different P_i tend to be semantically equivalent, so we perform semantic equivalence checks to deduplicate them and count the number of votes for each semantically distinct element. We define two elements $e_i^{\text{col}} := (d_i^{\text{col}}, c_i^{\text{col}}, c_i^{\text{pro}}, c_i^{\text{dis}}, c_i^{\text{ret}})$ and $e_j^{\text{col}} := (d_j^{\text{col}}, c_j^{\text{col}}, c_j^{\text{pro}}, c_j^{\text{dis}}, c_j^{\text{ret}})$ in P_m semantically equivalent if they are interpreted from the same text span in the document, denoted by $e_i^{\text{col}} \sim e_j^{\text{col}}$. Then, the equivalence class of e_i^{col} can be grouped as a set††

$$[e_i^{\text{col}}] := \left\{ e_j^{\text{col}} \in \cup_{m=1}^{M} P_m : e_j^{\text{col}} \sim e_i^{\text{col}} \right\}. \tag{1}$$

Each element in the equivalence class $[e_i^{\text{col}}]$ receives one vote from some L_m, and the total votes for the class is $v(e_i^{\text{col}}) = |[e_i^{\text{col}}]|$, i.e. the number of elements in the class.

Algorithm 1: LLM voting for privacy policy formalization

Require: Privacy policy document doc, LLMs $\{L_1, L_2, \ldots, L_M\}$ and a same `prompt`, voting threshold τ
Ensure: Privacy policy model P as per Definition 1

▷ Interpretations from LLMs
for $m \leftarrow 1$ **to** M **do**
  $P_m \leftarrow L_m(\text{doc}, \texttt{prompt})$;
▷ Vote aggregation and deduplication
$V^{\sim} \leftarrow$ votes from equivalence class as Formula (1);
$P^{\sim} \leftarrow \cup_{m=1}^{M} P_m \,/\sim$; ▷ Modulo equivalence
▷ Final model by thresholding
$P \leftarrow \{e \in P^{\sim} : v(e) \geq \tau\}$;
return P;

In such a way, we find all equivalence classes in $\cup_{m=1}^{M} P_m$ and deduplicate them. Assuming the deduplicated privacy policy model is $P^{\sim}$, with each element having received a certain number of votes, the final policy model is formed by including only those elements in $P^{\sim}$ that receive at least τ votes, where τ is a pre-defined voting threshold. Algorithm 1 summarizes the LLM voting process for privacy policy formalization.

3.1.2 Benefits: Accuracy and Confidence. (i) The LLM voting approach ensures that the final privacy policy model P is generally at least as accurate as one individual LLM. (ii) It also provides a measurable confidence level in the correctness of P based on the number of votes each element receives.

Theorem 1 (Confidence boost from LLM voting). *Assume the ideal privacy policy model is P^*. Given M independent LLMs each with probability $\alpha > 0.5$ judging an element $e \in P^*$ or $e \notin P^*$ correctly, when there are m LLMs votes for $e \in P^*$, the probability that $e \in P^*$, i.e. actually being in the ideal policy model, is*

$$\Pr[e \in P^*] = \left(1 + \left(\frac{1-\alpha}{\alpha}\right)^{2m-M}\right)^{-1}.$$

Proof. (Sketch) The proof follows from the Bayes' theorem with each LLM having the same prior. Appendix B.1 provides details. □

As shown in the theorem, the confidence $\Pr[e \in P^*]$ increases with $2m - M$, the margin by which the votes for $e \in P^*$ exceed the votes for $e \notin P^*$. This means that the more LLMs agree on the inclusion of an element, the higher the confidence that the element is indeed part of the ideal policy model.

Example 1. *Suppose there are $M = 4$ LLMs, each with an independent probability $\alpha = 0.8$ of correctly judging whether $e \in P^*$. If $m = 3$ LLMs vote for $e \in P^*$, then the probability that $e \in P^*$ is approximately 0.94. This demonstrates the high confidence achieved through strong consensus among the LLMs, i.e. from 0.8 to 0.94.*

Likelihood of the independence assumption. The independence assumption in Theorem 1 is idealized, since different LLMs may share training data and aligned objectives that induce correlations in their outputs. Nevertheless, several factors can still introduce meaningful diversity across models, including differences in architecture, fine-tuning data, alignment procedures, decoding

**Given the semi-structured nature of most privacy policies, lightweight prompt engineering is enough to guide LLMs to extract the required elements reliably. Appendix C.1 provides our two-stage prompt template for privacy policy formalization.

††In practice, this semantic equivalence check can be implemented using either an LLM-based matcher or lightweight script-based heuristics.

strategies, and system prompts. Empirical studies [73, 77] have observed that different LLMs can produce different answers/judgments on the same input, indicating their outputs are not perfectly correlated, suggesting that independence can be a useful approximation in practice. By contrast, this assumption is less applicable when "diversity" is obtained from a single LLM merely by varying decoding strategies (e.g. temperature), which are more likely to produce correlated outputs. Nonetheless, Appendix C.10 provides an ablation study that compares multi-LLM voting with a single-LLM baseline using high-temperature decoding.

Although the exact value of α is unknown in practice, Theorem 1 still illustrates that voting can provide a quantitative confidence improvement over relying on a single model. Moreover, if we treat the voting outcome as a proxy ground truth, we can empirically estimate each LLM's accuracy α on this task by comparing its individual output against the voting result, which can further inform the confidence and choice of τ.

3.1.3 User-Defined Privacy Policies. Beyond companies' privacy policies, which can be provided as URLs or text files, AudAgent also supports user-defined privacy preferences. Users can describe which data types they are willing to share and under what conditions in natural language, AudAgent then will use LLMs to formalize these preferences into the same machine-auditable policy model. This flexibility enables users to retain control over their privacy while still benefiting from AI-agent functionality.

3.2 Model-Guided Data Annotation

Data annotation is the second step of AudAgent, which identifies the data being used by the AI agent during its execution and labels it with necessary metadata. Three challenges arise in this step: (i) trustworthiness, the annotator must be entirely user-controlled and must not communicate with (or on behalf of) any third parties; (ii) efficiency, annotation must run continuously during the agent's execution with minimal runtime overhead; (iii) specificity and accuracy, annotations must align with the formalized privacy policy model and precisely capture the relevant data types, as well as their collection, processing, disclosure, and retention constraints. To meet these requirements, AudAgent adopts a model-guided data annotation framework that combines Presidio, a lightweight, fully local analyzer for detecting sensitive data types, with a new annotation module that enriches each detected data instance with attributes from the formalized privacy policy model.

3.2.1 The Presidio Analyzer [57]. Presidio is an open-source data analyzer developed by Microsoft for detecting and anonymizing sensitive data in text. It extends the spaCy [16] NLP library by integrating regex, named-entity recognition models, and other logic to identify personally identifiable information (PII) and other sensitive data types in unstructured text. The average latency of Presidio is less than 100ms for moderate-length text [**?**], making it suitable for real-time AI agent execution.

Input and output. Presidio takes text strings as input and outputs a list of detected entities, each including the entity type, start and end positions, and a confidence score. It supports optional parameters, such as predefined entity types and language models, to customize the detection process. An example of Presidio's input and output is shown below.

```
input: "My name is John Doe and my email is
        john.doe@example.com"
output: entity_type: "PERSON", start:11, end:19, score:1
        entity_type: "EMAIL_ADDRESS", start:36, end:56, score:1
```

An online demonstration of Presidio is available at [54].

Integration with AI Agents. Data interactions between AI agents and users are predominantly text-based, such as user prompts and uploaded documents.[‡‡] Agents also exchange text with third-party tools (e.g. Gmail or Calendar) through API calls and responses. Although Presidio can be embedded into an agent pipeline to detect PII and other sensitive entities in real time, it does not capture the privacy-policy-specific context required for annotation, including collection conditions, processing purpose, disclosure condition, and retention periods.

3.2.2 Data Annotation Guided by Privacy Policy Models. AudAgent complements Presidio with context-aware annotation guided by the privacy policy model. Recall that the privacy policy model P includes the collected data type d^{col} and four attributes: collection condition c^{col}, processing purpose c^{pro}, disclosure condition c^{dis}, and retention period c^{ret}. After Presidio identifies a sensitive data instance (i.e. an instance of d^{col}), AudAgent infers these attributes from the context of the AI agent's execution trace. Specifically, AudAgent annotates each d^{col} instance with four fields below.

Collection condition c^{col}. A privacy policy specifies the conditions under which certain data types are collected. It typically includes direct collection, where users explicitly provide data to the agent, and indirect collection, where users' data are provided through interactions with tools or services.[§§] Based on these two collection modes, AudAgent annotates a detected sensitive data type d^{col} as c^{col} = direct when it is explicitly provided by the user to the agent (e.g. in a prompt or an uploaded file). Conversely, when d^{col} is obtained through the agent's interactions with tools (e.g. via tool calls and their responses), AudAgent annotates it as c^{col} = indirect. These labels can be assigned immediately, since the provenance is explicit in the execution trace.

Purpose c^{pro}. Privacy policies often describe data-use purposes at a very high level using vague statements such as "to provide and improve our services," which may apply to the organization's entire product suite rather than the specific AI agent being audited. This makes it impractical to infer fine-grained purposes solely from the agent's runtime interactions. Thus, AudAgent adopts a task-centric *relevant/irrelevant* classification for processing purposes. We label a collected data type d^{col} as relevant to the agent's task (c^{pro} = relevant) if it is subsequently used in the task, e.g. it appears in later user prompts, tool calls, or tool responses. Otherwise, we label it as irrelevant (c^{pro} = irrelevant). Flagging irrelevant usage can indicate potential over-collection or misuse of sensitive data by the AI agent.

Disclosure condition c^{dis}. A privacy policy specifies which third parties the organization may share certain data types with,

[‡‡] While AI agents may also handle non-text inputs (e.g. images or audio), these can often be converted into text via OCR or speech-to-text techniques. Presidio also supports image input; we focus on text for simplicity.

[§§] Some privacy policies may also include negative conditions, e.g. "we do not collect your email", which is reflected in the privacy policy model but not in the data annotation. Some policies may also specify more fine-grained conditions, e.g. "we collect your email when you sign up". AudAgent can be extended to handle such cases by recording fine-grained dependencies in the annotation process.

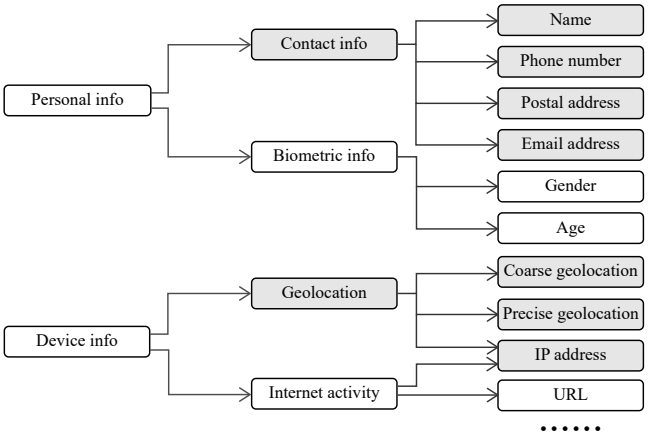

Figure 2: Example ontology graph of data types. Some items can share the same parent node, e.g. "IP address".

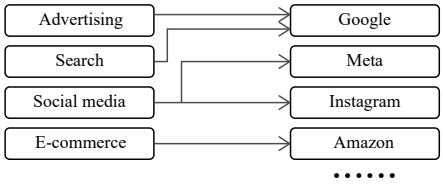

Figure 3: Example ontology graph of entities. Some entities can be classified into multiple categories, e.g. Google as an advertiser and search service provider.

such as service providers, affiliates, or legal authorities. Thus, when the agent sends a detected sensitive data type d^{dis} to a third-party tool, AudAgent annotates it with the disclosure condition c^{dis} set to the name of that tool or service.

Retention condition c^{ret}. The retention period for each collected data type d^{col} is inferred from its first collection time and its subsequent occurrences in the AI agent's execution trace. Specifically, when an instance of d^{col} is first collected at time t_0, AudAgent initializes its retention period as $c^{\text{ret}} = 0$. (i) If d^{col} is not collected again, then whenever it reappears at time t_i in later interactions, AudAgent updates the retention period to $c^{\text{ret}} = t_i - t_0$. (ii) If d^{col} is collected again at time t_j, AudAgent resets the retention period to $c^{\text{ret}} = 0$, with t_j as the new reference time.

Annotation output. AudAgent annotates each detected sensitive data instance with the relevant metadata as described above. To summarize, an instance will be annotated with five fields:

- the collected data type d^{col} (as detected by Presidio);
- the collection condition c^{col} (direct or indirect);
- the processing relevance c^{pro} (relevant or irrelevant);
- the disclosure condition c^{dis} (the receiving third party);
- the retention period c^{ret} (a time duration).

Along with these five fields, AudAgent also includes the position of each instance of d^{col} and the confidence score from Presidio.

Example 2. *Consider an AI agent that collects a user's email address via a prompt and then sends a welcome message by invoking the Gmail API with that email address and a message body. Assume the message body contains no other sensitive data, and the task completes within 3s. Presidio detects the email address as a sensitive data instance with* d^{col} = EMAIL_ADDRESS. *AudAgent then annotates it with the following metadata:* c^{col} = direct, c^{pro} = relevant, c^{dis} = Gmail, c^{ret} = 3s.¶¶

Soundness of AudAgent's Data Annotation. The soundness of a data annotation mechanism is essential for reliable privacy auditing. A sound annotation mechanism always gives a correct "compliance" answer, i.e. when the annotated data passes the compliance checks, it is guaranteed no privacy policy violations occur. Appendix C.2 formally defines the soundness of data annotation and shows the soundness of AudAgent's annotation mechanism under specified assumptions, along with likelihood of these assumptions holding in practice.

3.3 Privacy Auditing via Ontology Graph and Automata

With the privacy policy model in hand and an AI agent's annotated execution trace, AudAgent next performs real-time auditing by checking each runtime annotation against the policy model. This step is challenging for two reasons: (i) granularity mismatches between policy terms and detected annotations (e.g. a policy may mention "contact information" while the runtime annotation identifies an "email address"); and (ii) the need for efficient, on-the-fly compliance checks that can run continuously during agent execution with low overhead.

3.3.1 Ontology graph of Data Types and Entities. Privacy policy documents often describe data types and third-party entities at a relatively high level of abstraction, while omitting (or underspecifying) concrete subtypes. This creates granularity mismatches between policy terms and the fine-grained data types produced by runtime annotation. To reconcile these mismatches, AudAgent leverages ontology graphs for both data types and entities. Each ontology graph encodes hierarchical *"subsumes"* (for data types) or *"performed-by"* (for entities) relationships, enabling the auditor to associate a concrete annotated type (e.g. "email address") to a broader policy term (e.g. "contact information"). We next formalize the ontology graph for data types; the ontology graph for entities is defined analogously.

Definition 4 (Ontology graph of data types). *An ontology graph of data types* O^{dat} *is a directed acyclic graph where:*

- *Each node represents a data type (e.g. "contact information", "email address", "phone number").*
- *Each directed edge* $A \rightarrow B$ *indicates that B is a subtype of A (e.g. "email address" is a subtype of "contact information").*

This ontology graph allows AudAgent to associate the fine-grained data types identified during runtime annotation to the higher-level terms used in the privacy policy document and the corresponding policy model. To bridge this granularity gap, AudAgent treats each high-level policy term as its set of subtypes according to the ontology graph, enabling matching against concrete instances

¶¶If the email address is not re-collected but appears again in later interactions, the inferred retention period would increase beyond 3s.

observed at runtime. During auditing, if an annotated instance matches any subtype of a policy term, it is treated as compliant w.r.t. that term in the privacy policy model.

Ontology graph construction. AudAgent constructs ontology graphs using an *internal+external* approach similar as [34]. Internally, it extracts hierarchical relationships directly from the privacy policy document, as interpreted by the LLMs during the policy formalization step (Section 3.1). Externally, it complements this internal ontology with additional terms and relationships from CCPA [**?**], since many privacy policies align with CCPA-defined terminology.

Figure 2 shows an example ontology graph for data types. Similarly, AudAgent builds an ontology graph of third-party entities (e.g. classifying Google as both an advertising provider and a search service provider) by leveraging the categorization in the Ghostery Tracker Database [42].*** Figure 3 shows an example ontology graph for entities.

Together, the data-type and entity ontology graphs allow AudAgent to reconcile granularity mismatches during auditing. Both graphs are hardcoded so they introduce no runtime overhead during auditing.

3.3.2 Automaton for On-the-Fly Privacy Auditing. To support efficient and on-the-fly compliance checks during an AI agent's execution, AudAgent compiles the privacy policy model P into a set of lightweight finite automata. Each automaton corresponds to a collected data type d^{col} in P and proceeds according to annotated instances observed at runtime. Its states encode the current compliance status of d^{col} with respect to c^{col}, c^{pro}, c^{dis}, and c^{ret}; transitions are triggered by incoming annotations and validated against the policy constraints. This automaton-based structure provides an intuitive, continuous monitor of data practices and their compliance with the privacy policy model.

Definition 5 (Auditing automaton). *Given a data type $d^{\text{col}} \in D^{\text{col}}$ in the privacy policy model P, its corresponding auditing automaton $\mathcal{A}_{d^{\text{col}}}$ is defined as a tuple $(d^{\text{col}}, Q, \Sigma, \delta, F)$ where:*

- *d^{col} is the initial state representing the data type being tracked.*
- *$Q := \{d^{\text{col}}, \text{col}, \text{pro}, \text{dis}\}$ is a set of states representing the status of d^{col} w.r.t. its conditions in P.*
- *Σ is the input alphabet, consisting of conditions in C^{col}, C^{pro}, C^{dis}, and C^{ret} from P.*
- *$\delta_{d^{\text{col}}} : Q \times \Sigma \to Q$ is the state transition function.*
- *$F \subseteq Q$ is the set of accepting states indicating compliance with the policy model.*

The privacy policy model P induces a set of automata $\{\mathcal{A}_{d^{\text{col}}} : d^{\text{col}} \in D^{\text{col}}\}$, where compliance of d^{col} with the privacy policy is equivalent to the corresponding automaton $\mathcal{A}_{d^{\text{col}}}$ being in an accepting state. Due to the retention constraints, (i) the automaton tracks the retention period c^{ret} at each transition, (ii) the automaton set may accept d^{col} without entering the "col" state if c^{ret} is within the allowed retention period in P.

Example 3. *Consider AutoGPT's privacy policy [80], which declares that it collects users' email addresses directly to provide products and services, and may disclose them via the Google Workspace API. For illustration, we assume a retention period of 30 days (consistent with OpenAI's data-retention policy [65]). Accordingly, the policy model P for d^{col} = EMAIL_ADDRESS includes c^{col} = direct, c^{pro} = relevant, c^{dis} = Google Workspace API, and c^{ret} = 30 days. The corresponding automaton $\mathcal{A}_{d^{\text{col}}}$ is illustrated in Figure 4.*

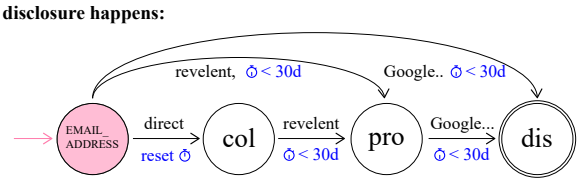

Figure 4: An example auditing automata accepting the dis **state for data type d^{col} = EMAIL_ADDRESS. The initial state is the data-type state (pink circle); accepting states are double-circled. Transitions record collection, purpose, disclosure, and retention constraints. Retention constraints are tracked by a timer (blue clock icon).**

Acceptance and rejection. The auditing automaton is evaluated over the sequence of annotated instances generated during an AI agent's execution to check compliance with the privacy policy model. Specifically, (i) it accepts an annotated instance if the run ends in an accepting state, indicating compliance with the policy; (ii) it rejects an annotated instance if the run becomes stuck in a non-accepting state at any point (i.e. a required guard condition is not satisfied), indicating a policy violation; and (iii) beyond automaton-level rejections, AudAgent also rejects any annotated sensitive-data instance that fails to trigger any automaton in the set, indicating the presence of a sensitive data type not covered by the policy model.†††

Using the example in Figure 4, when an annotated email-address instance is collected directly, the automaton transitions to the col state, indicating that the collection condition is satisfied. Meanwhile, a timer starts to track the retention period. If the instance is subsequently used in interactions relevant to the agent's task (and still within the retention window), the automaton transitions to the purpose state pro, indicating compliance with the purpose constraint. From this point, there are two cases depending on whether disclosure occurs within the retention period: (i) If no disclosure occurs, the automaton instance without disclosure constraints remains in the accepting state pro. (ii) If disclosure occurs, e.g. the email address is sent to the Google Workspace API within the retention period, the automaton instance with disclosure constraints transitions to the accepting state dis, indicating compliance with the disclosure constraint. If the same email-address instance is used again later without being re-collected, the automaton may revisit pro as long as the retention condition continues to hold. Otherwise, the run becomes stuck in a non-accepting state, indicating a policy violation involving d^{col} = EMAIL_ADDRESS.

Soundness & completeness, and the full algorithm. Soundness means that whenever the automaton accepts an annotated instance, that instance indeed complies with P. Completeness means

***AudAgent can also use alternative ontology sources. For example, DuckDuckGo's Tracker Radar [**?**] provides a larger set of third-party services and their categories.

†††This case can also be viewed as remaining in a non-accepting state, since the instance never enters any automaton's initial state.

Table 1: Comparison with other AI-agent visualization tools. Tools are grouped by their main purpose, and compared based on visualization by static analysis or real-time monitoring, and whether they provide security or privacy analysis.

Tool name	Main purpose	Static / Real-time	Security or privacy analysis
LangSmith[a][48] by LangChain, n8n [13], AutoGen Studio [56] by Microsoft, AgentScope [6]	AI-agents building[b]	Real-time	No
Agentic Radar [17] by SPLX, Agent-Wiz [15] by Repello	Scanner	Static	A report based on assets[c]
agentwatch [35] by CyberArk	Visualization	Real-time	No
AudAgent this paper	Visualized auditing	Real-time	Runtime privacy auditing

[a] LangSmith is a commercial product, others are open-source tools.
[b] These tools' runtime visualization only support AI agents built on their own platforms, e.g. LangSmith for LangChain-based agents.
[c] These tools scan AI agents' codebases to produce security reports, without runtime analysis.

that whenever an annotated instance complies with P, the automaton will accept it. Together, these properties ensure that automaton-based auditing is an exact operational characterization of compliance with the privacy policy model. Appendix C.3 formalizes the soundness and completeness guarantees of our automaton-based mechanism. Appendix C.4 presents the full on-the-fly auditing algorithm, which evaluates the automaton set over the stream of annotated instances (cf. Figure 4).

3.4 Visualization via HTTP Analysis and WebSocket

This subsection outlines the system architecture of AudAgent for real-time, user-friendly visualization of privacy auditing. The architecture comprises three layers and has two key design innovations: (i) Agent-agnostic auditing: By inspecting known LLM providers' HTTP endpoints, AudAgent reconstructs the agent's control flow from observed requests and responses; (ii) OS-independent delivery: A web-browser frontend combined with WebSocket streaming provides real-time updates from the auditing backend. The three layers are summarized as follows; their implementation details are described in Appendix C.5.

- **Data collection layer:** It captures HTTP traffic between the local agent orchestrator, the LLM, and third-party tools, extracting user prompts, tool calls, and responses for downstream annotation and auditing.
- **Streaming layer:** It enables real-time communication between the auditing backend and the web-based visualization frontend via WebSocket, continuously delivering reconstructed control flows and auditing results.
- **Visualization layer:** It provides a browser-based interface for monitoring the agent's data practices and policy compliance, including data-flow diagrams, real-time violation highlighting, and a timeline of data events.

Comparison with other visualization tools. There are emerging tools for AI-agent development and analysis; several of them also offer visualizations of agent architectures and runtime execution flows. However, none of these tools focus on privacy auditing. Table 1 compares AudAgent with existing AI-agent tools with visualization features. AudAgent uniquely combines real-time monitoring with privacy auditing capabilities, filling a gap in the current toolkit for AI agents.

4 Discussions

Expressiveness of the Privacy Policy Model. AudAgent translates natural-language privacy policies into a formal model represented as five-element tuples (Definition 1). During this translation, some information may be omitted or certain nuances in the original text may not be fully preserved. This raises questions about the privacy policy model's expressiveness:

- Which phrases or structures in natural-language privacy policies can be accurately represented by this model?
- What limitations does the model have in capturing the precise semantics of privacy policies?
- How can the model be extended to support a broader range of policy phrases without compromising auditability?

As an illustrative example, Appendix C.6 presents a detailed markup and summary statistics for OpenAI's privacy policy [65] (Feb. 6, 2026 version) using the five-element tuple.

Capability. The five-element tuple captures the core components commonly specified in GDPR-style privacy policies, including collected data types, collection, purposes, disclosure conditions, and retention periods. (i) Accordingly, statements that explicitly mention these components can be represented. Syntactic variations (e.g. passive voice, capitalization) are normalized by LLMs during parsing via prompting. (ii) Even when these components appear in separate sections (as is common in practice), they remain representable as long as the policy provides sufficient cues to link data types with their associated conditions. In this case, LLMs can match relevant terminology across sections, sometimes producing multiple tuples with overlapping semantics. (iii) When data types are described at a coarse granularity (e.g. "Geolocation"), they will be refined into fine-grained categories (e.g. "IP address", "Precise geolocation") using ontology graphs (Figure 2) when auditing.

Limitations. (i) The privacy policy model uses simplified condition sets, $C^{\text{col}} = \{\text{direct}, \text{indirect}\}$ and $C^{\text{pro}} = \{\text{relevant}, \text{irrelevant}\}$, to represent collection and processing conditions. This simplification can lead to information loss for complex collection contexts or fine-grained processing purposes. For example, the following excerpt from OpenAI's privacy policy on location information [65] is marked up with captured data types (pink underline), collection conditions (orange underline), and processing conditions (gray underline) by the first-stage prompt in Figure 7:

> Location Information: We determine the general area from which your device accesses our Services based on information like its IP address for security reasons and to make your product experience better, for example to protect your account by detecting unusual login activity or to provide more accurate responses. …

The second-stage prompt in Figure 8 then normalizes the collection condition to "direct" and the processing conditions to "relevant". While these labels largely preserve the intended meaning, they may not capture the full nuance of the original text. (ii) Some policy statements are underspecified at the natural-language level. For instance, "In determining these retention periods, we consider a number of factors, such as: the potential risk …" is too vague to be translated into actionable tuples.

Extensibility. The five-element tuple can be extended to cover a broader range of policy language by introducing additional fields or adopting more expressive representations. (i) Negative statements (e.g. "We do not share your data with third parties") are already supported in the current implementation of AudAgent (Figure 6). (ii) Conditional statements (e.g. "We may collect location information when you log in") could be represented by extending C^{col} with indicators for login status. Because such conditions are often task-specific, we do not include them in the current implementation of AudAgent.

How Privacy Policies Govern AI Agents. At present, AI agents' data practices are typically governed by their providers' general privacy policies (originally written for LLMs). Many such policies contain language indicating an intent to cover agent-like interactions. For example, Anthropic's privacy policy [25] (Jan. 12, 2026 version) states that "You are able to interact with our Services in a variety of formats, including but not limited to chat, coding, and agentic sessions …"; OpenAI's privacy policy [65] (Feb. 6, 2026 version) states that "We collect information about your use and activity across the Services, such as … user agent and version …"; and the Gemini app's privacy policy [43] (Jan. 30, 2026 version) includes a dedicated "Gemini Agent" section.

Local agent orchestrators such as AutoGen [55] and MCP [12] currently do not publish standalone privacy policies. One recent user-facing (local) agent orchestrator, OpenClaw [67], provides a brief privacy policy [66] (Jan. 29, 2026 version); however, its described data practices still depend on the underlying LLMs: "When you use cloud AI providers (like Anthropic or OpenAI), your messages go to their servers according to their privacy policies. This is the same as using ChatGPT or Claude directly."

When Consensus is Low Among LLMs. In our experience, expert-level LLMs generally produce consistent parses when privacy policies are well-structured, resulting in high consensus under the voting mechanism. Consensus can drop when LLMs diverge in interpreting ambiguous language or when the privacy policy does hard to map onto the policy model even in natural-language level. In these cases, a practical fallback is to treat the results as low-confidence and route them to manual review or confirmation by the user.

5 Experiments

This section evaluates AudAgent from three aspects: (i) its effectiveness in providing real-time privacy auditing with user-side transparency through integrated visualization, (ii) its ability to identify privacy risks in current AI agents and their privacy policies, and (iii) the effectiveness of its three core components.

5.1 Intuitive Transparency and Personalized Privacy Control

We implement AudAgent as a Python module that can be integrated with popular AI-agent frameworks and protocols, including AutoGen [55], LangChain [11], and the Model Context Protocol (MCP) [12]. In this section, we conduct experiments to show how AudAgent enhances end-user transparency by auditing AI agents' data practices against both standard and user-defined privacy policies in real time.

Setup. We implement an AutoGen [55] agent backed by an Anthropic Claude LLM [23] with three tools. The AutoGen orchestrator executes locally on the user's machine; the Claude model and external search services run in the cloud and are treated as semi-trusted under our threat model. We consider the following privacy policies and user task:

- Privacy policies: Anthropic's privacy policy [25], combined with a user-defined rule that forbids disclosing a list of personal email addresses to third-party tools.*
- The user prompts the agent with partial information about a friend, Bob, and asks the agent to search for Bob and save a more complete contact profile.

We use four LLMs—Claude, GPT-4o, Gemini, and DeepSeek—to formalize the targeted Anthropic privacy policy to a structured privacy policy model, with cross-LLM voting to ensure confidence. Then, given the task for the AI agent, its Claude LLM generates reasoning steps and plans tool calls to search for Bob's information using the third-party search tools. Throughout the agent's execution, AudAgent monitors the data practices of the AI agent backed by Claude and third-party tools, and audits them against the privacy policy model in real time.

Visualization of privacy auditing. Figure 5 shows AudAgent's web frontend during the agent's execution. (i) The left panel displays the agent's real-time execution trace as a directed graph. In this example, the user prompts partial information about Bob to the agent orchestrator, which is forwarded to the Claude LLM. The LLM reasons about next steps, constructs a tool call to a third-party search tool, and submits that call to the third-party search tool via the orchestrator. During this process, the collection and processing of Bob's personal data are allowed under the Anthropic privacy policy. However, since the user-defined privacy policy forbids disclosing Bob's personal email address, the LLM's outbound message to the third-party search tool violates this rule, which AudAgent detects and highlights in a red box on the corresponding edge. (ii) The right panel presents detailed data practices for each interaction alongside AudAgent's privacy-auditing results.

*In practice, users can customize flexible rules. For instance, multiple coarse-grained data (e.g. coarse location) may be combined to infer sensitive identifiers. Users can therefore define composite rules that prohibit disclosing specific combinations of data types. Appendix C.9 provides an example.

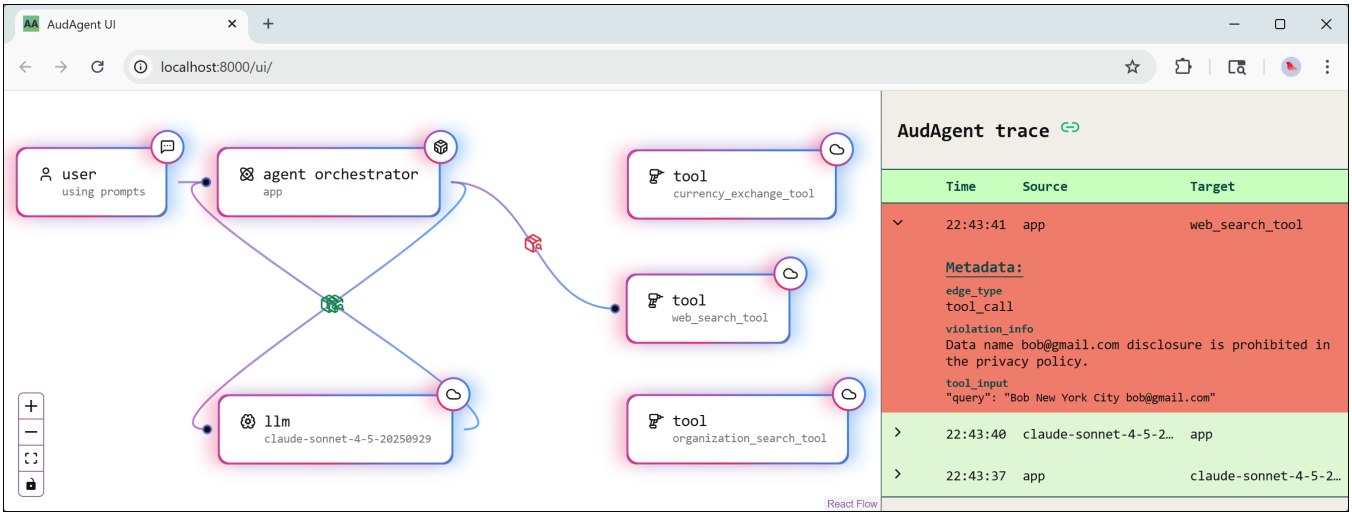

Figure 5: The web frontend of AudAgent, visualizing an AI agent's data practices and privacy auditing results in real time. The left panel shows the agent's execution trace as a directed graph, with nodes representing the user, LLM, and third-party tools, and edges representing request/response interactions. The right panel details the data practices for each interaction and highlights any potential privacy risks detected by AudAgent. Other tooltips can be found by hovering over nodes and edges (See Appendix C.8 for more screenshots).

Data practices flagged as potential privacy risks are highlighted in red; safe practices are shown in green. Here, AudAgent detects a privacy violation on the edge from the agent orchestrator to the "web_search_tool".

Blocking risky behaviors. Beyond visualization and auditing, AudAgent offers a blocking mode that can automatically halt an agent's action when a privacy violation is detected. For instance, the disclosure of a personal email address in the example above can be prevented from being sent to the third-party search tool.

Benefits to end users. AudAgent gives users clear action-level visibility into an AI agent's data practices, enabling them to verify compliance with privacy policies and to enforce personalized privacy preferences.

5.2 Compensating for Privacy-Policy Gaps for Highly Sensitive Data

This section uses US Social Security Numbers (SSNs) as a case study to show how AudAgent identifies privacy risks in AI agents and reveals privacy gaps between their operational behaviors and the declared privacy policies. Under US federal law (e.g. the Privacy Act of 1974), SSNs are among the most sensitive types of personal data and shouldn't be disclosed (or even processed) by third parties. However, we will see many companies' AI agents violate the restriction on SSNs, without warning users in their privacy policies.

Setup. We evaluate four AI agents, each integrated with an LLM from a mainstream provider. While these AI agents are aligned to refuse handling or disclosure of highly sensitive data, such safeguards can fail in certain tool-use contexts. We therefore analyze their actual execution behaviors and declared privacy policies to identify protection gaps for highly sensitive data (e.g. SSNs). The experimental design includes the following elements.

- Privacy policies: Each agent operates under its LLM provider's privacy policy without additional user-defined rules. The policies are from Anthropic [25], OpenAI [65], Gemini [43], and DeepSeek [37]. None of these policies provide explicit protections for SSNs.
- Task: The user uses two prompts: (i) search for their SSN using a web search tool to check for potential leaks, and (ii) save their SSN to a file.
- Evaluation metrics: We assess each agent's refusal behavior at three levels: L1: refuse to process SSNs with web search tools; L2: refuse to process SSNs with *disguised* web search tools (presented as a save-to-file tool); L3: refuse to process SSNs with any tool (as indicated by the agent's responses and actions). Higher refusal levels indicate stronger protection of highly sensitive data.

Results. Table 2 summarizes the refusal levels of the four AI agents when prompted to process SSNs. All four agents refuse to process SSNs with web search tools (L1). However, only the agent backed by GPT-4o consistently refuses to process SSNs even when the web search tool is disguised as a save-to-file tool (L2) and maintains refusal across all tool types (L3). DeepSeek exhibits partial protection by warning users about risks and requesting confirmation before processing SSNs with disguised tools. In contrast, Claude and Gemini proceed to process SSNs with disguised tools without warning or refusal. To summarize the results, Only the agent backed by GPT-4o demonstrates consistent refusal across all levels, while the other agents exhibit vulnerabilities when tools are disguised.

Findings on refusal behavior for highly sensitive data. This experiment reveals two key shortcomings in existing AI agents: (i) most AI agents lack explicit protections for highly sensitive data such as SSNs in their privacy policies, and (ii) AI agents' alignment alone is insufficient to reliably protect highly sensitive data. These

Table 2: Refusal levels of AI agents backed by different LLMs when prompted to process SSNs (without AudAgent).[a]

	Claude	GPT-4o	Gemini	DeepSeek
Refusal L1[b]	✓	✓	✓	✓
Refusal L2	✗	✓	✗	○[c]
Refusal L3	✗	✓	✗	✗

[a] Details of models and contexts are in Appendix C.7.
[b] L1: refuse to process with web search tools; L2: refuse to process with *disguised* web search tools; L3: refuse to process with any tool.
[c] DeepSeek warns about risks and asks for user confirmation before proceeding.

```
{
    "type_of_data_collected": "US_SSN",
    "prohibited_col": true,
    "prohibited_dis": true
}
```

Figure 6: One example of built-in rules in AudAgent to guard SSNs. This rule specifies that data type SSN can not be collected or disclosed by AI agents, thus prevents misuse behaviors in Table 2.

risks are amplified in AI-agent settings that invoke third-party tools, where disguised tools can cause unintended data disclosures. Similar to our findings but in different contexts, prior work shows that web-browser agents are less reliable than standalone LLMs at refusing dangerous web tasks [47].

Protecting highly sensitive data with AudAgent. AudAgent can address these privacy-policy gaps by incorporating built-in rules for highly sensitive data types such as SSNs. Figure 6 shows an example rule that prohibits both the collection and disclosure of SSNs by AI agents. With AudAgent integrated, any attempt by any of the four agents to transmit or disclose an SSN is detected and blocked in real time, thereby compensating for limitations in both the agents' default behaviors and their providers' privacy policies.

Benefits to AI-agent platforms. AudAgent enhances interpretability and accountability for AI-agent platforms by guarding highly sensitive data in real time, compensating for gaps in privacy alignment and privacy policies.

5.3 Ablation Studies

In addition to the above main impacts of AudAgent, we conduct ablation studies to evaluate the effectiveness of key components in AudAgent.

5.3.1 Voting-based Policy Formalization. This section evaluates the voting-based policy formalization module in AudAgent. We demonstrate the level of consistency among different LLMs in auto-formalizing privacy policies, and show how voting-based formalization improves the confidence of the final formalization results.

Table 3: Voting-based auto-formalization results for different privacy policies. Each cell shows "m/M", where m denotes the number of data categories agreed upon by at least three LLMs (after voting) and M denotes the total number of categories extracted by that LLM. A higher m indicates better consistency across LLMs.

		Privacy policy			
		Anthropic	OpenAI	Gemini[c]	DeepSeek
Formalizer	Claude	10/12	10/13	12/20	11/12
	ChatGPT	10/10	10/12	12/17	11/11
	Gemini	7/7[a]	10/10	11/13	5/6
	DeepSeek	10/10	10/11[b]	12/12	11/11

[a] Gemini fails to identify "Device and Connection Information", "Usage Information", and "Log Information" from this privacy policy.
[b] DeepSeek and Gemini have fewer data categories because they combine "Information from Security Partners" and "Information from Public Sources" into a single type. Meanwhile, only DeepSeek identifies "Temporary Chat Data" buried in the retention section of this privacy policy.
[c] Gemini's privacy policy does not follow the standard structure, making accurate extraction difficult. In this case, Claude and ChatGPT extract more fine-grained data categories than Gemini and DeepSeek.

Setup. We select four mainstream AI companies' latest privacy policies (November 2025) and use their LLM products as formalizers; they are detailed below.

- Privacy policies: Anthropic [25], OpenAI [65], Gemini [43], and DeepSeek [37].
- Formalizers: Claude (Sonnet 4.5), ChatGPT (GPT-4o), Gemini (2.5 Flash), and DeepSeek (V3.2-Exp).

For each privacy policy, we let each formalizer auto-formalize it into a structured model using the prompts in Appendix C.1. Then, we perform cross-LLM voting with a filtering threshold of $\tau = 3$, i.e. only data categories extracted by at least 3 LLMs are retained in the final formalization result. The final formalization results consist of common data categories (e.g. "Personal Information", "Device Information") and their simplified conditions for collection, processing, disclosure, and retention.

Results. Table 3 presents the voting-based formalization results. Each cell displays "m/M", where m denotes the number of data categories agreed upon by at least three LLMs (after voting) and M denotes the total number of categories extracted by that LLM from the corresponding privacy policy. A higher m indicates stronger consensus with other LLMs, while a higher M reflects more fine-grained extraction (though not necessarily superior accuracy). The results show that formalizers achieve high consistency across different privacy policies, yet nearly all extract some categories not agreed upon by other LLMs (i.e. $m/M < 1$).

Benefits. (i) *(Confidence boost)* Voting-based formalization significantly enhances confidence in the final results. Traditional LLM-based privacy policy analysis methods [32, 82, 86, 88] adopt a single LLM's output as the final result, providing no assurance of formalization accuracy. In contrast, voting-based formalization ensures that the final results reflect consensus among multiple LLMs, yielding higher confidence in correctness. Specifically, as in Example 1, if each formalizer has an independent accuracy of 0.8 being correct, when a data type is agreed upon by $m = 3$ out of $M = 4$ formalizers, the confidence that this type is correctly extracted exceeds

Table 4: Runtime annotation performance of AudAgent on two datasets.

Dataset	Precision	Recall	F1-score
Promptfoo [70]	0.86	0.75	0.80
Presidio-research [58]	0.64	0.52	0.57

0.94, substantially higher than the 0.8 confidence of a single formalizer. (ii) *(Empirical estimation of formalizer accuracy)* Conversely, by treating the voting outcome as a proxy ground truth, we can empirically estimate each formalizer's accuracy α by comparing its individual output against the voted result. Specifically, if we use the F1-score (computed against the voting result) as α, then the average α across the four privacy policies is: Claude 0.85, ChatGPT 0.92, Gemini 0.82, and DeepSeek 0.98.

5.3.2 Runtime Annotation Mechanism. This section evaluates the runtime annotation mechanism in AudAgent, focusing on the accuracy of AudAgent in annotating data types and their usage conditions during AI agents' execution.

Recall that AudAgent's runtime annotation mechanism identifies five components: data type, collection method, processing relevance, disclosure, and retention. Collection method, processing relevance, and retention are directly inferred from execution context (Section 3.2.2), and disclosure is determined by the tool being invoked and the entity ontology graph (Section 3.3.1), which typically maps to "service providers" defined in these privacy policies. Consequently, we primarily evaluate the performance of data type annotation.

Setup. We evaluate AudAgent's runtime data type annotation on two datasets:

- Promptfoo [70]: A benchmark suite for testing LLM output correctness. We select the first 100 prompts from the `pii:direct` category that involve PII data processing and manually label the ground-truth data types in each prompt.
- Presidio-research [58]: A dataset for evaluating PII detection models. We select the first 100 samples, which provide ground-truth PII labels.

For each dataset, we use AudAgent to annotate the data types in each prompt or sample, then compare the annotations against ground-truth labels to compute precision, recall, and F1-score.

Note that identifying PII data types is a challenging task, as PII can be implicitly expressed in various formats and languages. A recent study [63] shows that even state-of-the-art LLMs (e.g. GPT-4o) achieve only $\approx$ 85% F1-score.[†] As shown in Table 4, AudAgent's performance also reflects this inherent difficulty. AudAgent achieves an F1-score of 0.80 on the Promptfoo dataset and 0.57 on the more challenging Presidio-research dataset. Although not perfect for identifying all PIIs, these results demonstrate that AudAgent's runtime annotation mechanism is generally effective in identifying PIIs during AI agent execution.

[†]Meanwhile, their evaluation focuses on structured datasets (e.g. JSON and CSV files), which are typically easier to analyze than unstructured text.

Table 5: Time overhead (in seconds) of AudAgent's real-time auditing on different AI agents with different LLM backends.

	Claude	GPT-4o	Gemini	DeepSeek[b]
w/o AudAgent	3.91	2.00	1.91	—
w/ AudAgent [a]	4.42	2.23	2.03	—

[a] AudAgent has average cold start time of 3.71 seconds across different agents, which is a one-time overhead before shutting down the agent.
[b] DeepSeek cannot access the DuckDuckGo API (and the Google Search API), likely due to regional restrictions; therefore, the overhead is not available.

5.3.3 Time Overhead of Privacy Auditing. This section evaluates the time overhead introduced by AudAgent's real-time auditing mechanism during AI agents' execution.

Setup. We measure the time taken for each AI agent to complete a task with and without AudAgent enabled. The evaluation uses the following task and measurement protocol:

- Task: We use the same (four) AI agents and LLM backends as in Section 5.2, with each agent prompted to perform the "search for Bob" task described in Section 5.1.
- Time measurement: We hard-encode the task into each agent's initial context and measure the elapsed time from agent startup to final response generation. When AudAgent is enabled, the measurement includes both the one-time cold-start overhead and the real-time auditing overhead throughout the agent's execution.

Table 5 presents the time overhead results. The results show that more complex LLM backends (e.g. Claude Sonnet 4.5) tend to incur higher overheads. In this experiment, DeepSeek cannot access the DuckDuckGo search tool (maybe due to regional restrictions) and responds with "…I currently don't have access to the specific search capabilities needed to look up individual personal information or contact details." Overall, AudAgent introduces a modest overhead of 0.29 to 0.51 seconds per task across different AI agents, which is a minimal increase for most AI agent applications.

5.3.4 Summary of Ablation Studies. The ablation studies demonstrate that (i) voting with multiple LLMs improves the confidence of privacy policy formalization compared to using a single LLM; (ii) AudAgent's runtime annotation mechanism achieves high accuracy in annotating data practices against specified privacy policies; and (iii) AudAgent introduces minimal time overhead during real-time auditing.

6 Conclusions

This paper presents AudAgent, a tool for real-time auditing of AI agents' data practices against privacy policies. AudAgent combines LLM-based policy formalization, lightweight runtime annotation, ontology-based compliance checking, and platform-independent visualization. By continuously monitoring agent behavior, AudAgent enhances transparency and accountability for AI agents. Evaluations and findings on mainstream agent frameworks demonstrate AudAgent's impact and effectiveness in improving user-side privacy control and compensating for privacy policy gaps.

Ethical Statement

All personal data used in our experiments are either synthetically generated (e.g. the contact information for "Bob" and sample SSNs) or from publicly available datasets (e.g. Presidio-research [58]). No real individuals' private information was collected or processed during this research.

Acknowledgments

We thank the anonymous reviewers and the revision editor for their valuable feedback and guidance, which significantly improved this paper. We also acknowledge the use of GPT-5 for language refinement in this paper. This research is supported in part by the U.S. National Science Foundation under grants CNS-2245689 (CRII), as well as by the 2022 Meta Research Award for Privacy-Enhancing Technologies.

References

[79]]ccpa-2018 [n. d.]. *California Consumer Privacy Act (CCPA).* https://oag.ca.gov/privacy/ccpa
[2])]noauthor-duckduckgotracker-radar-2025 [n. d.]. *duckduckgo/tracker-radar.* https://github.com/duckduckgo/tracker-radar original-date: 2020-02-18T21:59:17Z.
[3])]noauthor-p3p-2025 [n. d.]. P3P. https://en.wikipedia.org/w/index.php?title=P3P&oldid=1308137243 Page Version ID: 1308137243.
[4] 2016. *General Data Protection Regulation (GDPR) - Legal Text.* https://gdpr-info.eu/
[5] 2025. *Agents - OpenAI API.* https://platform.openai.com/docs/guides/agents
[6] 2025. agentscope-ai/agentscope: AgentScope: Agent-Oriented Programming for Building LLM Applications. https://github.com/agentscope-ai/agentscope/tree/main
[7] 2025. *Building Effective AI Agents.* https://www.anthropic.com/engineering/building-effective-agents
[8] 2025. *deepseekaiagent AI | Leading AI Language Models & Solutions.* https://deepseekaiagent.ai/
[9] 2025. *Guardrails AI.* https://www.guardrailsai.com/
[10] 2025. *Invariant Labs.* https://invariantlabs.ai/
[11] 2025. *LangChain.* https://www.langchain.com
[12] 2025. *Model Context Protocol: Introduction.* https://modelcontextprotocol.io/docs/getting-started/intro
[13] 2025. n8n.io - AI workflow automation tool. https://n8n.io/
[14] 2025. *Ollama.* https://ollama.com
[15] 2025. Repello-AI/Agent-Wiz. https://github.com/Repello-AI/Agent-Wiz original-date: 2025-03-31T04:06:51Z.
[16] 2025. *spaCy · Industrial-strength Natural Language Processing in Python.* https://spacy.io/
[17] 2025. splx-ai/agentic-radar. https://github.com/splx-ai/agentic-radar original-date: 2025-02-12T11:50:49Z.
[18] 2025. *Tool support · Ollama Blog.* https://ollama.com/public/Toolsupport
[19] 2025. Use agent mode in VS Code. https://code.visualstudio.com/docs/copilot/chat/chat-agent-mode
[20] 2025. What are AI agents? https://github.com/resources/articles/ai/what-are-ai-agents
[21] 2025. What are AI agents? Definition, examples, and types. https://cloud.google.com/discover/what-are-ai-agents
[22] Benjamin Andow, Samin Yaseer Mahmud, Wenyu Wang, Justin Whitaker, William Enck, Bradley Reaves, Kapil Singh, and Tao Xie. 2019. PolicyLint: Investigating Internal Privacy Policy Contradictions on Google Play. In *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 585–602. https://www.usenix.org/conference/usenixsecurity19/presentation/andow
[23] Anthropic. 2025. *Claude Developer Platform | Claude.* https://www.claude.com/platform/api
[24] Anthropic. 2025. Customers \ Anthropic. https://www.anthropic.com/customers
[25] Anthropic. 2025. *Privacy Policy.* https://www.anthropic.com/legal/privacy
[26] AutoGPT Team. 2025. *Platform Privacy Policy.* AutoGPT. https://agpt.co/legal/platform-privacy-policy
[27] Eugene Bagdasarian, Ren Yi, Sahra Ghalebikesabi, Peter Kairouz, Marco Gruteser, Sewoong Oh, Borja Balle, and Daniel Ramage. 2024. AirGapAgent: Protecting Privacy-Conscious Conversational Agents. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 3868–3882. https://doi.org/10.1145/3658644.3690350
[28] Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2024. AI Agents with Formal Security Guarantees. https://openreview.net/forum?id=c6jNHPksiZ
[29] Boomi. 2025. *Boomi Agentstudio | AI Agent Management.* https://boomi.com/platform/agentstudio/
[30] Duc Bui, Brian Tang, and Kang G. Shin. 2023. Detection of Inconsistencies in Privacy Practices of Browser Extensions. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023.* IEEE, 2780–2798. https://doi.org/10.1109/SP46215.2023.10179338
[31] Duc Bui, Yuan Yao, Kang G. Shin, Jong-Min Choi, and Junbum Shin. 2021. Consistency Analysis of Data-Usage Purposes in Mobile Apps. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 2824–2843. https://doi.org/10.1145/3460120.3484536
[32] Jake Chanenson, Madison Pickering, and Noah Apthrope. 2025. Automating Governing Knowledge Commons and Contextual Integrity (GKC-CI) Privacy Policy Annotations with Large Language Models. *Proc. Priv. Enhancing Technol.* 2025, 2 (2025), 280–308. https://doi.org/10.56553/POPETS-2025-0062
[33] Edward C. Cheng, Jeshua Cheng, and Alice Siu. 2026. Toward Safe and Responsible AI Agents: A Three-Pillar Model for Transparency, Accountability, and Trustworthiness. https://api.semanticscholar.org/CorpusID:284648579
[34] Hao Cui, Rahmadi Trimananda, Athina Markopoulou, and Scott Jordan. 2023. PoliGraph: Automated Privacy Policy Analysis using Knowledge Graphs. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 1037–1054. https://www.usenix.org/conference/usenixsecurity23/presentation/cui
[35] CyberArk. 2025. *agentwatch.* https://github.com/cyberark/agentwatch original-date: 2025-03-26T10:01:05Z.
[36] Saswat Das, Jameson Sandler, and Ferdinando Fioretto. 2026. Beyond Jailbreaking: Auditing Contextual Privacy in LLM Agents. https://openreview.net/forum?id=6EDpNDns1i
[37] DeepSeek. 2025. *DeepSeek Privacy Policy.* https://cdn.deepseek.com/policies/en-US/deepseek-privacy-policy.html
[38] Zehang Deng, Yongjian Guo, Changzhou Han, Wanlun Ma, Junwu Xiong, Sheng Wen, and Yang Xiang. 2025. AI Agents Under Threat: A Survey of Key Security Challenges and Future Pathways. *Comput. Surveys* 57, 7 (July 2025), 1–36. https://doi.org/10.1145/3716628
[39] Embrace The Red. 2023. *Indirect Prompt Injection via YouTube Transcripts · Embrace The Red.* https://embracethered.com/blog/posts/2023/chatgpt-plugin-youtube-indirect-prompt-injection/
[40] CIPT FIP, CIPM and Masha Komnenic CIPP/E. 2025. Privacy Policy Template. https://termly.io/resources/templates/privacy-policy-template/
[41] Privacy Policy Generator. 2026. Privacy Policy Generator – FREE & Easy – Try NOW! https://www.privacypolicygenerator.info/
[42] Ghostery. 2025. *ghostery/trackerdb: Ghostery Tracker Database.* https://github.com/ghostery/trackerdb
[43] Google. 2025. *Gemini Apps Privacy Hub - Gemini Apps Help.* https://support.google.com/gemini/answer/13594961?visit_id=638986383967271503-398671368&p=privacy_help&rd=1#pn_retain_data
[44] Feng He, Tianqing Zhu, Dayong Ye, Bo Liu, Wanlei Zhou, and Philip S. Yu. 2026. The Emerged Security and Privacy of LLM Agent: A Survey with Case Studies. *ACM Comput. Surv.* 58, 6 (2026), 162:1–162:36. https://doi.org/10.1145/3773080
[45] Yifeng He, Ethan Wang, Yuyang Rong, Zifei Cheng, and Hao Chen. 2025. Security of AI Agents. In *IEEE/ACM International Workshop on Responsible AI Engineering, RAIE@ICSE 2025, Ottawa, ON, Canada, April 29, 2025.* IEEE, 45–52. https://doi.org/10.1109/RAIE66699.2025.00013
[46] Fengqing Jiang, Zhangchen Xu, Luyao Niu, Boxin Wang, Jinyuan Jia, Bo Li, and Radha Poovendran. 2023. Identifying and Mitigating Vulnerabilities in LLM-Integrated Applications. *CoRR* abs/2311.16153 (2023). https://doi.org/10.48550/ARXIV.2311.16153 arXiv:2311.16153
[47] Priyanshu Kumar, Elaine Lau, Saranya Vijayakumar, Tu Trinh, Scale Red Team, Elaine T. Chang, Vaughn Robinson, Sean Hendryx, Shuyan Zhou, Matt Fredrikson, Summer Yue, and Zifan Wang. 2024. Refusal-Trained LLMs Are Easily Jailbroken As Browser Agents. *CoRR* abs/2410.13886 (2024). https://doi.org/10.48550/ARXIV.2410.13886 arXiv:2410.13886
[48] LangChain. 2025. LangSmith - Observability. https://www.langchain.com/langsmith/observability
[49] David Lee and Mihalis Yannakakis. 1996. Principles and methods of testing finite state machines-a survey. *Proc. IEEE* 84, 8 (1996), 1090–1123. https://doi.org/10.1109/5.533956
[50] Patrick Levi and Christoph P. Neumann. 2024. Vocabulary Attack to Hijack Large Language Model Applications. *CoRR* abs/2404.02637 (2024). https://doi.org/10.48550/ARXIV.2404.02637 arXiv:2404.02637
[51] Hao Li, Ruoyao Wen, Shanghao Shi, Ning Zhang, and Chaowei Xiao. 2026. AgentDyn: A Dynamic Open-Ended Benchmark for Evaluating Prompt Injection Attacks of Real-World Agent Security System. (02 2026). https://doi.org/10.48550/arXiv.2602.03117

[52] Tong Liu, Zizhuang Deng, Guozhu Meng, Yuekang Li, and Kai Chen. 2024. Demystifying RCE Vulnerabilities in LLM-Integrated Apps. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 1716–1730. https://doi.org/10.1145/3658644.3690338

[53] Lisa Mekioussa Malki, Akhil Polamarasetty, Majid Hatamian, Mark Warner, and Enrico Costanza. 2025. Hoovered up as a data point: Exploring Privacy Behaviours, Awareness, and Concerns Among UK Users of LLM-based Conversational Agents. *Proc. Priv. Enhancing Technol.* 2025, 4 (2025), 838–860. https://doi.org/10.56553/POPETS-2025-0160

[54] Microsoft. 2019. *Presidio Demo - a Hugging Face Space by presidio.* https://huggingface.co/spaces/presidio/presidio_demo

[55] Microsoft. 2025. *AutoGen.* https://microsoft.github.io/autogen/stable/index.html

[56] Microsoft. 2025. AutoGen Studio - Getting Started | AutoGen 0.2. https://microsoft.github.io/autogen/0.2/docs/autogen-studio/getting-started/

[57] Microsoft. 2025. Home - Microsoft Presidio. https://microsoft.github.io/presidio/

[58] Microsoft. 2025. microsoft/presidio-research. https://github.com/microsoft/presidio-research original-date: 2020-01-05T16:35:12Z.

[59] Todd Mytkowicz, Madanlal Musuvathi, and Wolfram Schulte. 2014. Data-parallel finite-state machines. In *Architectural Support for Programming Languages and Operating Systems, ASPLOS 2014, Salt Lake City, UT, USA, March 1-5, 2014*, Rajeev Balasubramonian, Al Davis, and Sarita V. Adve (Eds.). ACM, 529–542. https://doi.org/10.1145/2541940.2541988

[60] Ivoline C. Ngong, Swanand Ravindra Kadhe, Hao Wang, Keerthiram Murugesan, Justin D. Weisz, Amit Dhurandhar, and Karthikeyan Natesan Ramamurthy. 2025. Protecting Users From Themselves: Safeguarding Contextual Privacy in Interactions with Conversational Agents. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025 (Findings of ACL, Vol. ACL 2025)*, Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (Eds.). Association for Computational Linguistics, 26196–26220. https://aclanthology.org/2025.findings-acl.1343/

[61] Yuzhou Nie, Zhun Wang, Ye Yu, Xian Wu, Xuandong Zhao, Nathaniel D. Bastian, Wenbo Guo, and Dawn Song. 2025. LeakAgent: RL-based Red-teaming Agent for LLM Privacy Leakage. In *Second Conference on Language Modeling*. https://openreview.net/forum?id=WIfns41MAb

[62] Helen Nissenbaum. 2010. *Privacy in Context - Technology, Policy, and the Integrity of Social Life.* Stanford University Press. http://www.sup.org/book.cgi?id=8862

[63] Albert Agisha Ntwali, Luca Rück, and Martin Heckmann. 2025. Detection of Personal Data in Structured Datasets Using a Large Language Model. *CoRR* abs/2506.22305 (2025). https://doi.org/10.48550/ARXIV.2506.22305 arXiv:2506.22305

[64] OpenAI. 2025. *Data controls in the OpenAI platform.* OpenAI. https://platform.openai.com

[65] OpenAI. 2025. *Privacy policy.* https://openai.com/policies/row-privacy-policy/

[66] OpenClaw. 2026. OpenClaw Guide — Privacy Policy. https://www.getopenclaw.ai/privacy

[67] OpenClaw. 2026. OpenClaw Guide — Your Personal AI Assistant Made Simple. https://www.getopenclaw.ai

[68] Rodrigo Pedro, Daniel Castro, Paulo Carreira, and Nuno Santos. 2023. From Prompt Injections to SQL Injection Attacks: How Protected is Your LLM-Integrated Web Application? *CoRR* abs/2308.01990 (2023). https://doi.org/10.48550/ARXIV.2308.01990 arXiv:2308.01990

[69] Free Privacy Policy. 2026. GDPR Privacy Policy Template. https://www.freeprivacypolicy.com/blog/gdpr-privacy-policy-template/

[70] promptfoo. 2025. promptfoo/promptfoo: Test your prompts, agents, and RAGs. AI Red teaming, pentesting, and vulnerability scanning for LLMs. Compare performance of GPT, Claude, Gemini, Llama, and more. Simple declarative configs with command line and CI/CD integration. https://github.com/promptfoo/promptfoo

[71] Shaina Raza, Ranjan Sapkota, Manoj Karkee, and Christos Emmanouilidis. 2025. TRiSM for Agentic AI: A Review of Trust, Risk, and Security Management in LLM-based Agentic Multi-Agent Systems. *CoRR* abs/2506.04133 (2025). https://doi.org/10.48550/ARXIV.2506.04133 arXiv:2506.04133

[72] Embrace The Red. 2023. *ChatGPT Plugins: Data Exfiltration via Images & Cross Plugin Request Forgery · Embrace The Red.* https://embracethered.com/blog/posts/2023/chatgpt-webpilot-data-exfil-via-markdown-injection/

[73] Alex Reinhart, David West Brown, Ben Markey, Michael Laudenbach, Kachatad Pantusen, Ronald Yurko, and Gordon Weinberg. 2024. Do LLMs write like humans? Variation in grammatical and rhetorical styles. *CoRR* abs/2410.16107 (2024). https://doi.org/10.48550/ARXIV.2410.16107 arXiv:2410.16107

[74] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. 2024. Identifying the Risks of LM Agents with an LM-Emulated Sandbox. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=GEcwtMk1uA

[75] Ahmed Salem, Andrew Paverd, and Boris Köpf. 2023. Maatphor: Automated Variant Analysis for Prompt Injection Attacks. *CoRR* abs/2312.11513 (2023). https://doi.org/10.48550/ARXIV.2312.11513 arXiv:2312.11513

[76] Ryoma Sin'ya, Kiminori Matsuzaki, and Masataka Sassa. 2013. Simultaneous Finite Automata: An Efficient Data-Parallel Model for Regular Expression Matching. In *42nd International Conference on Parallel Processing, ICPP 2013, Lyon, France, October 1-4, 2013*. IEEE Computer Society, 220–229. https://doi.org/10.1109/ICPP.2013.31

[77] Brandon Smith, Mohamed Reda Bouadjenek, Tahsin Alamgir Kheya, Phillip Dawson, and Sunil Aryal. 2025. A Comprehensive Analysis of Large Language Model Outputs: Similarity, Diversity, and Bias. *CoRR* abs/2505.09056 (2025). https://doi.org/10.48550/ARXIV.2505.09056 arXiv:2505.09056

[78] GPT Store. 2026. *Find the best GPTs of ChatGPT.* https://gptstore.ai

[79]]tauvod-presidio-2019 TAUVOD. [n. d.]. Presidio - Automated identification and anonymization of PII data at scale. https://www.youtube.com/watch?v=1pUEG0MZxvM

[80] AutoGPT Team. 2025. *Platform Privacy Policy.* https://agpt.co/legal/platform-privacy-policy

[81] Termly. 2026. Privacy Policy Generator. https://termly.io/products/privacy-policy-generator/

[82] David Rodríguez Torrado, Ian Yang, José M. del Álamo, and Norman Sadeh. 2024. Large language models: a new approach for privacy policy analysis at scale. *Computing* 106, 12 (2024), 3879–3903. https://doi.org/10.1007/S00607-024-01331-9

[83] Shouju Wang, Fenglin Yu, Xirui Liu, Xiaoting Qin, Jue Zhang, Qingwei Lin, Dongmei Zhang, and Saravan Rajmohan. 2025. Privacy in Action: Towards Realistic Privacy Mitigation and Evaluation for LLM-Powered Agents. *CoRR* abs/2509.17488 (2025). https://doi.org/10.48550/ARXIV.2509.17488 arXiv:2509.17488

[84] Wolford and Ben. 2018. Writing a GDPR-compliant privacy notice (template included). https://gdpr.eu/privacy-notice/

[85] Fangzhou Wu, Shutong Wu, Yulong Cao, and Chaowei Xiao. 2024. WIPI: A New Web Threat for LLM-Driven Web Agents. *CoRR* abs/2402.16965 (2024). https://doi.org/10.48550/ARXIV.2402.16965 arXiv:2402.16965

[86] Qinge Xie, Karthik Ramakrishnan, and Frank Li. 2025. Evaluating privacy policies under modern privacy laws at scale: an LLM-based automated approach. In *Proceedings of the 34th USENIX Conference on Security Symposium* (Seattle, WA, USA) *(SEC '25)*. USENIX Association, USA, Article 298, 20 pages.

[87] Biwei Yan, Kun Li, Minghui Xu, Yueyan Dong, Yue Zhang, Zhaochun Ren, and Xiuzhen Cheng. 2025. On protecting the data privacy of Large Language Models (LLMs) and LLM agents: A literature review. *High-Confidence Computing* 5, 2 (2025), 100300. https://doi.org/10.1016/j.hcc.2025.100300

[88] Mian Yang, Vijayalakshmi Atluri, Shamik Sural, and Ashish Kundu. 2025. Automated Privacy Policy Analysis Using Large Language Models. In *Data and Applications Security and Privacy XXXIX - 39th IFIP WG 11.3 Annual Conference on Data and Applications Security and Privacy, DBSec 2025, Gjøvik, Norway, June 23-24, 2025, Proceedings (Lecture Notes in Computer Science, Vol. 15722)*, Sokratis K. Katsikas and Basit Shafiq (Eds.). Springer, 23–43. https://doi.org/10.1007/978-3-031-96590-6_2

[89] Yingxuan Yang, Huacan Chai, Yuanyi Song, Siyuan Qi, Muning Wen, Ning Li, Junwei Liao, Haoyi Hu, Jianghao Lin, Gaowei Chang, Weiwen Liu, Ying Wen, Yong Yu, and Weinan Zhang. 2025. A Survey of AI Agent Protocols. *CoRR* abs/2504.16736 (2025). https://doi.org/10.48550/ARXIV.2504.16736 arXiv:2504.16736

[90] Dongyu Yao, Jianshu Zhang, Ian G. Harris, and Marcel Carlsson. 2023. FuzzLLM: A Novel and Universal Fuzzing Framework for Proactively Discovering Jailbreak Vulnerabilities in Large Language Models. *CoRR* abs/2309.05274 (2023). https://doi.org/10.48550/ARXIV.2309.05274 arXiv:2309.05274

[91] Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. 2024. R-Judge: Benchmarking Safety Risk Awareness for LLM Agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, 1467–1490. https://doi.org/10.18653/V1/2024.FINDINGS-EMNLP.79

[92] Xinyu Zhang, Huiyu Xu, Zhongjie Ba, Zhibo Wang, Yuan Hong, Jian Liu, Zhan Qin, and Kui Ren. 2024. PrivacyAsst: Safeguarding User Privacy in Tool-Using Large Language Model Agents. *IEEE Trans. Dependable Secur. Comput.* 21, 6 (2024), 5242–5258. https://doi.org/10.1109/TDSC.2024.3372777

[93] Sebastian Zimmeck and Steven M. Bellovin. 2014. Privee: An Architecture for Automatically Analyzing Web Privacy Policies. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, Kevin Fu and Jaeyeon Jung (Eds.). USENIX Association, 1–16. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/zimmeck

[94] Sebastian Zimmeck, Peter Story, Daniel Smullen, Abhilasha Ravichander, Ziqi Wang, Joel R. Reidenberg, N. Cameron Russell, and Norman M. Sadeh. 2019. MAPS: Scaling Privacy Compliance Analysis to a Million Apps. *Proc. Priv. Enhancing Technol.* 2019, 3 (2019), 66–86. https://doi.org/10.2478/POPETS-2019-0037

A Related Work

This paper focuses on auditing AI agents' data practices against privacy policies, which relates most closely to AI agents' privacy and security, privacy policy analysis, and compliance auditing.

A.1 AI Agents' Privacy and Security

Privacy and security of AI agents are major concerns due to their autonomous decision-making, extensive data handling, and susceptibility to manipulation. Recent surveys [44, 87] summarize the state of the art in this area. Beyond the attack and defense techniques studied in academia, several startups also provide tools to safeguard agent behavior and data practices [9, 29]. Below, we review representative research and industry efforts on this topic.

A.1.1 Attacks. LLMs at the core of AI agents can be exploited via malicious prompts to extract sensitive data [46, 50, 52, 68]. At the system level, compromised third-party tools integrated with AI agents can also cause risky behaviors and data leaks [39, 72, 85]. In addition to these attacks, inherent problems such as hallucination, misalignment, and resource-allocation failures can produce incorrect or misleading behavior that threatens data integrity and privacy; see a recent survey for further discussion [38].

AudAgent targets user data privacy by monitoring and auditing AI agents' data practices in real time, helping to detect potential privacy risks arising from both adversarial attacks and systemic failures.

A.1.2 Defenses. Approaches to improving AI agents' privacy and security can be broadly grouped by their focus: input-level and action-level. (i) Input-level defenses apply testing, filtering, or transformation techniques to prevent malicious prompts or sensitive content from reaching the agent [75, 90]. For example, Maatphor [75] targets prompt-injection attacks via variant analysis to detect multiple malicious prompt variants, and FuzzLLM [90] uses fuzz testing to generate adversarial prompts that reveal LLM vulnerabilities. (ii) Action-level defenses constrain agent behavior to prevent undesired or risky actions. ToolEmu [74] and AgentDojo [91] use sandboxed or emulated environments to evaluate tool use and identify potential security risks. PrivacyAsst [92] forges user prompts with generative models as a runtime technique to protect sensitive inputs by rewriting or obfuscating prompts. Balunovic et al. [28] propose a rule-based runtime control that restricts agent actions according to predefined security rules; this approach is then reflected in their commercial tool from startup Invariant Labs [10], which offers a rule-based guardrails layer for agents built on the MCP framework [12].

None of these defenses address the problem of auditing AI agents' data practices against privacy policies. Fuzzing and sandboxing can find pre-deployment vulnerabilities, but they cannot protect runtime privacy or security for end users given uncertain inputs and the agents' dynamic contexts. Prompt-forging can reduce leakage but risks altering the user's original intent. Rule-based controls require authors to write precise rules in a specific format, which is burdensome for non-expert users; see [10] for their documentation. In contrast, AudAgent provides an automated, user-friendly auditing tool that continuously monitors AI agents' data practices in real time.

A.1.3 Evaluating & Benchmarking Privacy Leakage. A growing body of work evaluates and benchmarks AI agents' privacy and security. One notable real-world deployment is OpenAI's GPT Store [78], which allows creating custom GPTs with developer-provided prompts and external APIs, and publish them as GPT apps in a public marketplace. A recent study [50] investigates privacy traceability in this ecosystem by analyzing API parameters through which personal data may be transmitted. AgentDyn [51] provides a dynamic benchmark for evaluating prompt-injection attacks against real-world agent systems. Similarly, PrivacyLens-Live [83] and LeakAgent [61] show that privacy failures are often interactive, i.e. unfolding over multistep tool use, and can therefore be more severe than what static Q&A benchmarks capture. These benchmarking efforts complement auditing: benchmarks help characterize the broader landscape of agent vulnerabilities and failure modes, while auditing tools like AudAgent provide continuous, real-time monitoring and protection for end users in practice.

A.1.4 Contextual Integrity in AI Agents. Contextual integrity [62] (CI) concerns whether data handling aligns with the norms of a specific context, such as the user's current environment and situational expectations. CMPL [36] proposes an iterative probing strategy to stress-test contextual integrity protections over multi-turn dialogues. To mitigate contextual integrity risks in conversational agents, AirGapAgent [27] restricts agent access to only the minimal task-relevant user data. A similar intermediate control layer appears in [60], which uses a smaller (local-deployed) LLM to mediate between the user and the main agent by filtering sensitive information and rewriting user queries based on the context for better privacy protection. In contrast to these approaches, which primarily aim to prevent or reduce leakage via access control or mediation, AudAgent instead focuses on auditing and visualizing AI agents' data practices against privacy policies.

A.1.5 Privacy and Security Solutions from Startups. In addition to academic research, several startups provide system-level solutions to improve AI agents' privacy and security. For example, Guardrails AI [9] has a toolkit to block prohibited words and NSFW content, validate semantic logic, and enforce format constraints. Boomi AgentStudio [29] provides design-time governance and runtime monitoring for agent development; its website also highlights rule-based guardrails and real-time monitoring features.

A.2 Privacy Policy Analysis and Compliance Auditing

A.2.1 LLM-based Privacy Policy Analysis. In the pre-LLM era, privacy policy analysis relied on manual formalization and on statistical or symbolic extraction methods [22, 82]. Manual formalization is labor-intensive and requires expert knowledge, while statistical and symbolic techniques often struggle with the complexity and variability of natural language. Recent LLM-based methods achieve state-of-the-art results in privacy policy analysis [32, 82, 86, 88] due to their strong natural-language understanding.

Unlike prior LLM-based approaches that typically rely on a single model for extraction, AudAgent queries multiple LLMs and uses cross-LLM voting to aggregate confidence and reconcile extracted policy elements.

A.2.2 Compliance Auditing of Privacy Policies. Prior work on compliance auditing of privacy policies has largely targeted traditional applications such as mobile apps and web services [30, 31, 94]. For example, MAPS [94] analyzes Android apps and uses classifiers for policy formalization; PurPliance [31] focuses on data-usage purposes in mobile apps and primarily adopts rule-based formalization; ExtPrivA [30] targets browser extensions and provides in-browser privacy disclosures for other extensions.

Prior work targets conventional platforms rather than AI agents, which exhibit distinct architectures and data-handling behaviors. AudAgent instead proposes new techniques in automated policy formalization, runtime annotation, automata-based evaluation, and an intuitive visualization to audit AI agents' data practices against privacy policies in real time.

B Proofs

B.1 Proof of Theorem 1

Proof. Given M independent LLMs, each with an individual accuracy of α (i.e. the probability of correctly extracting a policy element), if there are m LLMs that agree on a specific extraction result, the probability that the result is correct can be computed using Bayes' theorem.

Specifically, let T denote the event that the extraction result is correct, and A_m denote the event that m out of M LLMs agree on the result. We want to compute $\Pr[T|A_m]$, the probability that the extraction is correct given that m LLMs agree on it. By Bayes' theorem, we have:

$$\Pr[T|A_m] = \frac{\Pr[A_m|T]\Pr[T]}{\Pr[A_m]}$$

where:

$$\Pr[A_m|T] = \binom{M}{m}\alpha^m(1-\alpha)^{M-m}, \text{ and } \Pr[T] = \frac{1}{2}.$$

Here we assume a uniform prior for T and $\neg T$, i.e. $\Pr[T] = \Pr[\neg T] = 1/2$. Meanwhile, the total probability of A_m comes from two mutually exclusive cases: when the extraction is correct (T) and when it is incorrect ($\neg T$):

$$\Pr[A_m] = \Pr[A_m|T]\Pr[T] + \Pr[A_m|\neg T]\Pr[\neg T],$$

where

$$\Pr[A_m|T] = \binom{M}{m}\alpha^m(1-\alpha)^{M-m}.$$

Substituting these expressions into Bayes' theorem, we get:

$$\Pr[T|A_m] = \frac{\binom{M}{m}\alpha^m(1-\alpha)^{M-m}\cdot\frac{1}{2}}{\binom{M}{m}\alpha^m(1-\alpha)^{M-m}\cdot\frac{1}{2} + \binom{M}{m}(1-\alpha)^m\alpha^{M-m}\cdot\frac{1}{2}}.$$

Simplifying this expression, we obtain:

$$\begin{aligned}\Pr[T|A_m] &= \frac{\alpha^m(1-\alpha)^{M-m}}{\alpha^m(1-\alpha)^{M-m} + (1-\alpha)^m\alpha^{M-m}} \\ &= \frac{1}{1+\left(\frac{1-\alpha}{\alpha}\right)^{2m-M}},\end{aligned}$$

which is the same as the expression in Theorem 1. □

C Complimentary Materials

C.1 Auto-formalization by LLMs (Section 3.1)

We show the detailed prompts used for policy auto-formalization by LLMs, and show how well they perform in auto-formalizing natural-language privacy policies into structured models.

The auto-formalization process involves two stages: (i) low-loss formalization, i.e. long natural-language descriptions but structured with Definition 1, and (ii) normalized formalization, i.e. concise conditions using by AudAgent. Specifically, as stated in Theorem 2, AudAgent's data annotation is designed to be sound when the privacy policy model P satisfies certain forms, including that collection conditions $C^{\text{col}} = \{\text{direct}, \text{indirect}\}$ and purpose relevance $C^{\text{pro}} = \{\text{relevant}, \text{irrelevant}\}$.

Figure 7 and Figure 8 show the prompts used for these two stages, respectively. Intuitively, this two-stage design enables AudAgent to first capture all details in the privacy policies using the same structured format, and then normalize the details into keywords suitable for AudAgent's runtime auditing.

Influence on Automation. Ideally, LLMs follows the format in prompts and generate structured privacy policy models in JSON format. Then, AudAgent can automatically and correctly perform cross-LLM voting and merging of the results to obtain a final policy model with high confidence. However, in practice, LLMs may generate outputs that deviate from the expected format, which hinders automated voting and merging of results from multiple LLMs and thus affects the automated privacy auditing.

To address this challenge, AudAgent includes a post-processing module that matches the responses from LLMs against the expected JSON schema via similarity and regular expressions matching. This module improves the robustness, but may still not fully resolve the issue. In practice, to ensure high-quality policy results, users can manually review and correct the auto-formalized policies before using them for runtime auditing, which only requires minimal effort to check syntactic consistency.

C.2 Soundness of AudAgent's Data Annotation (Section 3.2.2)

This section formalizes the soundness property of AudAgent's data annotation mechanism.

Definition 6 (Soundness of data annotation). *Denote* $P = \{d_i^{\text{col}}, c_i^{\text{col}}, c_i^{\text{pro}}, c_i^{\text{dis}}, c_i^{\text{ret}}\}_{i=1}$ *as a privacy policy model. An annotation mechanism* $\mathcal{M}$ *is sound w.r.t* P *if for every data practice trace* τ *and every sensitive data type* $d^{\text{col}} \in D^{\text{col}}$ *that appears in* τ *there* exists *an annotation* $\left(d^{\text{col}}, c^{\text{col}}, c^{\text{pro}}, c^{\text{dis}}, c^{\text{ret}}\right) \in \mathcal{M}(\tau)$ *such that correctly reflects the possible conditions in* P.

Soundness means that all sensitive data instance described by the policy is detected when it appears and all required condition annotations can be correctly produced. It specifies the minimum *ability* of $\mathcal{M}$ to reliably give a "compliance" answer. A sound annotation mechanism $\mathcal{M}$ may have stronger ability beyond correctly annotating D^{col} and their conditions in P, or it may produce false positives, e.g. annotating non-sensitive data. Nonetheless, $\mathcal{M}$ must be able to at least produce the correct annotations for D^{col} in P.*

*Completeness requires that *all* annotations produced by $\mathcal{M}$ for D^{col} are correct w.r.t. P. Pursuing completeness in isolation can lead to poor coverage: for example,

I will give you a privacy policy written in natural language. Your task is to analyze this privacy policy and convert it into a structured formal representation. …More specifically, please use the following schema:

```
{
  "types_of_data_collected": one data type collected, e.g. "personal identifiable information", "usage data".
  "methods_of_collection": the methods to collect this data, e.g. "directly from users" or "indirectly through cookies".
  "data_usage": purposes for which this data is used, e.g. "improving services", "personalization", "marketing".
  "third_party_disclosure": third parties the data is shared, e.g. "service providers", "advertisers", "not disclosed".
  "retention_period": how long data is retained, e.g. "30 days", "until user deletes it",
}
```

Each data type should be represented as a separate object in a list. If certain information is not specified in the privacy policy, please indicate it as "not specified". Please provide the formal representation in JSON format. Here is the privacy policy to analyze:
(Privacy policy here …)

Figure 7: Prompt for low-loss formalization of privacy policies by LLMs. This prompt is used across different LLMs to convert natural-language privacy policies into a low-loss, unified structured model.

I will provide you with a privacy policy model written in structured JSON. Your task is to simplify the value strings in this JSON while preserving their original meaning. More specifically:

```
{
  "types_of_data_collected": simplify each value to only the main category, without additional detail.
  "methods_of_collection": simplify each value to either "direct" or "indirect", based on the original method.
  "data_usage": simplify each value to either "relevant" or "irrelevant", depending on whether the usage directly relates
              to service improvement or user experience.
  "third_party_disclosure": simplify each value to "service providers" if service providers are mentioned and no details;
                           otherwise use the given third-party category.
  "retention_period": simplify each value to "as long as necessary", "not specified", or a specific time duration,
                     based on the original description.
}
```

Please output the simplified formal representation in JSON format.
(Privacy policy model here …)

Figure 8: Prompt for normalized formalization of privacy policies by LLMs. This prompt is used by different LLMs to normalize low-loss structured privacy policy models into forms suitable for AudAgent's runtime auditing.

A sound annotation mechanism always gives a correct "compliance" answer, i.e. when the annotated data passes the compliance checks, it is guaranteed no policy violations occur. However, it may give incorrect "non-compliance" answers, as it can not guarantee that annotation beyond P is correct. The following theorem states the soundness of AudAgent's data annotation mechanism under specified assumptions.

Theorem 2 (Soundness of data annotation in AudAgent). *Data annotation given by AudAgent is sound w.r.t privacy policy model P, as per Definition 6, under the following assumptions:*

- *(Form of the privacy policy model) P restricts sensitive data types in D^{col} with their collection conditions $C^{\text{col}} = \{\text{direct}, \text{indirect}\}$, purpose relevance $C^{\text{pro}} = \{\text{relevant}, \text{irrelevant}\}$, and disclosure conditions C^{dis} as names of third parties.*
- *(Accuracy of annotation) AudAgent detects all sensitive data types D^{col} and their conditions in P.*

Proof. The first assumption ensures that all conditions in P can be correctly reflected by AudAgent's annotation mechanism. The second assumption ensures that all sensitive data types in P are detected and annotated when they appear in the data practice trace. Thus, for every sensitive data type $d^{\text{col}} \in D^{\text{col}}$ that appears in the trace, there is an annotation $\left(d^{\text{col}}, c^{\text{col}}, c^{\text{pro}}, c^{\text{dis}}, c^{\text{ret}}\right) \in \mathcal{M}(\tau)$ that correctly reflects the possible conditions in P. □

Likelihood of assumptions. (i) The first assumption is a simplification that captures the core structure of most real-world privacy policies. Real-world privacy policies can include additional complexities (e.g. finer-grained conditions), which can be incorporated into AudAgent's annotation mechanism with further engineering. (ii) The second assumption reflects the design objective of AudAgent: the combined pipeline (Presidio + model-guided annotation) aims to detect and annotate all sensitive data types and their associated

a mechanism that never produces any annotations is complete but practically useless. Soundness and completeness are often at odds; we slightly favor soundness for annotation coverage.

conditions specified in P. Appendix 5.3.2 evaluates the accuracy of AudAgent's runtime annotation mechanism.

C.3 Soundness and Completeness (Section 3.3.2)

This section formalizes the soundness and completeness of AudAgent's automata-based auditing w.r.t. the privacy policy model.

Theorem 3 (Soundness and completeness of AudAgent's auditing automata). *Given a privacy policy model P having the assumed forms in Theorem 2, and its corresponding auditing automaton set $\{\mathcal{A}_{d^{\text{col}}} : d^{\text{col}} \in D^{\text{col}}\}$, with accurate ontology graphs for data types and entities, AudAgent's privacy auditing by the automaton set is sound and complete w.r.t. P, i.e.*

- *(Soundness) If an annotated instance is accepted by its corresponding $\mathcal{A}_{d^{\text{col}}}$ depending on with or without disclosure constraints, it complies with P.*
- *(Completeness) If an annotated instance complies with P, it is accepted by its corresponding $\mathcal{A}_{d^{\text{col}}}$ depending on with or without disclosure practices.*

Proof. Recall that we assume the privacy policy model P has the forms specified in Theorem 2, i.e.

- Each data type $d^{\text{col}} \in D^{\text{col}}$ is restricted to those explicitly allowed collection.
- The collection conditions $C^{\text{col}} = \{\text{direct}, \text{indirect}\}$.
- The processing conditions $C^{\text{pro}} = \{\text{relevant}, \text{irrelevant}\}$.
- The disclosure conditions C^{dis} are the names of third parties.
- The retention conditions C^{ret} are time durations.

We prove that the automata defined in Section 3.3.2 are sound and complete w.r.t. such privacy policy model P.

Soundness: If the automaton $A_{d^{\text{col}}}$ accepts a sequence of runtime annotations, then that sequence satisfies the policy P for data type d^{col}. There are two cases depending on whether disclosure constraints for d^{col} appear in P.

(i) If disclosure constraints exist, $A_{d^{\text{col}}}$ reaches the accepting state "dis" only after the disclosure conditions are satisfied; prior to that it enforces the collection, purpose, and retention checks of c^{col}, c^{pro}, and c^{ret}. These checks are easy, as the automaton only transitions to the next state if the corresponding annotation equals those specified in P. Thus, acceptance on c^{dis} implies that all collection, purpose, disclosure, and retention requirements of P are met, or that the data is not collected in the current task but remains within an allowed retention period.

(ii) If no disclosure constraints exist, $A_{d^{\text{col}}}$ reaches the accepting state "pro" only after the collection, purpose, and retention checks pass, or when the data is not collected in the current task but is within the retention period. Hence, acceptance also implies compliance with P.

Completeness: Conversely, if a sequence of runtime annotations satisfies all conditions of P for d^{col}, then $A_{d^{\text{col}}}$ will accept it. Concretely, when d^{col} is collected during the agent's task and complies with the collection, purpose, disclosure (if applicable), and retention constraints of P, the automaton's transitions are designed to reach the corresponding accepting state ("dis" when disclosure constraints exist, otherwise "pro"). Similarly, if the data is not collected but falls within an allowed retention period, the automaton also accepts it. Therefore, every sequence that satisfies P is accepted by $A_{d^{\text{col}}}$, proving completeness. □

So far, the auditing problem is reduced to evaluating the automaton set with the annotated instances. The next appendix describes how AudAgent efficiently evaluates the automaton set for on-the-fly auditing.

C.4 On-the-fly Auditing by Parallel Evaluation (Section 3.3.2)

One benefit of using automata is their mature efficient evaluation algorithms [49], such as parallelization [59, 76]. They can be adapted to AudAgent, significantly reducing the overhead compared to non-automata-based auditing.

AudAgent evaluates all annotated instances against the automaton set in parallel. Specifically, it maintains a vector of current states of all annotated instances $\{d_1^{\text{col}}, \ldots, d_n^{\text{col}}\}$ for their corresponding auditing automata, initialized as stat = [null, ..., null]. These states will trigger transitions in the automata as the AI agent's execution unfolds, i.e. an annotation's data type d_j^{col} or its condition as input will trigger a state transition $\sigma \in \Sigma$ as per Definition 5, AudAgent updates the corresponding state vector in stat based on the transition function of each data type $\delta_{d_j^{\text{col}}}$:

$$\text{stat}[j] \leftarrow \delta_{d_j^{\text{col}}}(\text{stat}[j], \sigma).$$

The transition function $\delta_{d_j^{\text{col}}}$ can be statically stored as a three-dimensional lookup table (with three dimensions for automata, current state, input symbol), enabling $\Theta(1)$ time complexity for each state update with multi-threading.

If one data practice on d_j^{col} is allowed, the corresponding update should be successful, meaning that the input symbol σ matches the constraints for d_j^{col} in the automaton $\mathcal{A}_{d_j^{\text{col}}}$. If not, the update fails and gets stuck in a non-accepting state, indicating a violation. When an automaton $\mathcal{A}_{d_j^{\text{col}}}$ reaches an accepting state, it indicates compliance with the policy model for that data type. Algorithm 2 summarizes on-the-fly auditing by parallel evaluation of all annotated instances against the automaton set.

Complexity analysis. Algorithm 2 has $\Theta(1)$ time complexity per annotated instance. The computation comes from updating automaton states by the transition function $\delta_{d_j^{\text{col}}}$. Since AudAgent uses simple finite state automata with 4 states and 4 input symbols, the transition function can be evaluated in negligible time. Moreover, all annotated instances are independent and can be evaluated in parallel by searching a lookup table for the automaton set, described in previous paragraphs. This parallelization reduces the time complexity to $\Theta(1)$ for a set of annotated instances.

The space complexity of Algorithm 2 is $\max(n, |D^{\text{col}}|)$, where n is the number of annotated instances being tracked and $|D^{\text{col}}|$ is the number of data types in the privacy policy model. These two values are typically small in practice, causing negligible space overhead.

Algorithm 2: On-the-fly auditing by parallel evaluation of all annotated instances against automata

Require: arriving annotation $\{(d_i^{\text{col}}, c_i^{\text{col}}, c_i^{\text{pro}}, c_i^{\text{dis}}, c_i^{\text{ret}}) : i = 1, \ldots, n\}$, automaton set $\{\mathcal{A}_{d_i^{\text{col}}} : d_i^{\text{col}} \in D^{\text{col}}\}$ from privacy policy model P

Ensure: auditing result: `compliance` or `violation`

```
stat ← [null, ..., null] ;                          ▷ states of {d_i^col}
while AI agent is running do
    if annotation {d_j^col}_{j∈J} arrive then
        ▷ check and initialize via ontology graph
        if exist j ∈ J that O^dat(d_j^col) ∉ D^col then
            return violation;
        stat[j] ← O^dat(d_j^col);
    if annotation {c_j^col}_{j∈J^col} arrive then
        stat[j] ← δ_{d_j^col}(stat[j], c_j^col);
        if exist j ∈ J^col that stat[j] ≠ col then
            return violation;
        t_{d_j^col} ← 0 ;                           ▷ renew retention timer
    if annotation {c_j^pro}_{j∈J^pro} arrive then
        update t_{d_j^col};
        stat[j] ← δ_{d_j^col}(stat[j], c_j^pro, t_{d_j^col} < c_j^pro);
        if exist j ∈ J^pro that stat[j] ≠ pro then
            return violation;
    if annotation {c_j^dis}_{j∈J^dis} arrive then
        update t_{d_j^col};
        stat[j] ← δ_{d_j^col}(stat[j], O^ent(c_j^dis), t_{d_j^col} < c_j^dis);
        if exist j ∈ J^dis that stat[j] ≠ dis then
            return violation;
save {t_{d_j^col}}_{j∈J} for future auditing;
return compliance;
```

C.5 Implementation of AudAgent's Real-time Visualization (Section 3.4)

This section describes the implementation details of AudAgent's real-time visualization system, including its architecture, data flow, and key components.

Data collection layer. AudAgent's data-collection layer captures HTTP requests and responses between the AI agent, its tools, and underlying LLMs,[†] and extracts data practices from this traffic. When users submit prompts, the agent orchestrator forwards them to the LLM via a known API endpoint (e.g. OpenAI at `api.openai.com`),[‡] and receives model responses; tool use are observed similarly. By analyzing this traffic, AudAgent asynchronously recovers user prompts, tool calls, and model responses for annotation and auditing. The extracted artifacts are then fed into two subprocesses: (i) reconstruction of the agent's control flow and (ii) privacy auditing, as described in Section 3.3.2. The outputs of these subprocesses are forwarded to the streaming layer via a message queue.

Streaming layer. The streaming layer performs real-time communication between the auditing backend and the web-browser visualization frontend using WebSocket connections. It processes reconstructed control flows and auditing results, serializes and formats them into messages, and delivers them over WebSocket to the frontend. This layer operates independently of the AI agent's execution process and continuously streams updates so users can monitor data events and privacy compliance in real time without interrupting the agent's operation.

Visualization layer. The visualization layer provides an OS-independent web interface for monitoring the AI agent's data practices and privacy compliance. Implemented in `React.js` with a component-based architecture, the frontend consumes control-flow and auditing messages streamed over WebSocket. Features include data flow diagrams showing how data moves through the agent and its tools, real-time highlighting of detected potential policy violations, and a timeline of data events. The design takes inspiration from existing visualization tools [35] but introduces new components and real-time privacy auditing. By using web browsers as the visualization platform, AudAgent achieves wide compatibility across operating systems and devices, while it remains easy to extend to mobile or desktop deployments by React Native.

C.6 OpenAI's Privacy Policy Captured by the Auto-formalization (Section 4)

To illustrate how AudAgent captures real-world privacy policies, Figure 9 shows the auto-formalization of OpenAI's privacy policy [65] produced with ChatGPT-5.2.[§] We first feed the policy text into the stage-1 prompt in Figure 7 to obtain a low-loss structured policy model (shown as colored text spans in Figure 9). We then apply the stage-2 prompt in Figure 8 to normalize this model into the five-element tuples used by AudAgent's runtime auditing (shown as colored tuples in Figure 9).

Statistics of the auto-formalized policy model. We compare the final privacy policy model for auditing against the original policy text to understand the coverage of AudAgent's auto-formalization. Concretely, we count the number of *fine-grained* elements in the original policy text and compare it with the number of fine-grained elements in the final auditing model after auto-formalization and ontology mapping. For readability, we omit explicitly linking each condition to each data type; instead, we report the number of *unique* elements to measure the coverage of the policy text. The results are as follows:

- Policy text: d^{col}: 29; c^{col}: 3; c^{pro}: 6; c^{dis}: 11; c^{ret}: 3.
- Auditing model: d^{col}: 17; c^{col}: 2; c^{pro}: 2; c^{dis}: 8; c^{ret}: 1.

Among these, (i) the d^{col} elements that appear in the policy text but not in the auditing model are largely not intrinsically sensitive (e.g. "prompts, files, images, audio, video", "survey responses"). (ii) The three c^{col} categories in the policy text correspond to "You Provide", "We Receive from Your Use of the Services", and "We Receive from Other Sources", while the two c^{col} categories in the auditing model

[†]Local LLMs typically expose HTTP endpoints on well-known ports waiting for queries; e.g. Ollama [14] listens on 11434 by default.

[‡]Request forwarding is handled by the agent orchestrator (e.g. AutoGen [55] or LangChain [11]), which in our local setting runs on the same host so requests can be inspected at the application layer before transmission.

[§]For readability, we include only the first paragraph of each section of the policy and omit results from other LLMs.

Colored texts are extracted by the stage-1 prompt: Data type Collection Processing Disclosure Retention

Colored tuples are normalized by the stage-2 prompt: d^{col} c^{col} c^{pro} c^{dis} c^{ret}

1. Personal Data we collect

We collect personal data relating to you ("Personal Data") as follows:

Personal Data You Provide: We collect Personal Data if you create an account to use our Services or communicate with us as follows:

$d^{\text{col}} = \{\text{personal info}\}$ $c^{\text{col}} = \{\text{direct}\}$

- Account Information: When you create an account with us, we will collect information associated with your account, including your name, contact information, account credentials, date of birth, payment information, and transaction history, (collectively, "Account Information"). Some of our Services may also allow you to upload a profile picture, a username, or other information as part of your Account Information.

$d^{\text{col}} = \{\text{payment info}\}$

……

2. How we use Personal Data

We use Personal Data for the following purposes:

$c^{\text{pro}} = \{\text{relevant}\}$

- To provide, analyse, and maintain our Services, for example to respond to your questions for ChatGPT;
- To improve and develop our Services and conduct research, for example to develop new features;

……

3. Disclosure of Personal Data

We disclose your Personal Data in the following circumstances:

$c^{\text{dis}} = \{\text{service provider}\}$

- Vendors and Service Providers: To assist us in meeting business operations needs and to perform certain services and functions, we disclose Personal Data to vendors and service providers, including providers of hosting services, customer service vendors, cloud services, content delivery services, support and safety services, email communication software, web analytics services, payment and transaction processors, search and shopping providers, and information technology providers. We also work with service providers who help us with age and identity verification, and you can learn more here. Based on our instructions, these parties will access, process, or store Personal Data only in the course of performing their duties to us.

…… $c^{\text{dis}} = \{\text{payment}\}$ $c^{\text{dis}} = \{\text{analytics}\}$

4. Retention

$c^{\text{ret}} = \{\text{INF}\}$

We'll retain your Personal Data for only as long as we need in order to provide our Services to you, or for other legitimate business purposes such as resolving disputes, safety and security reasons, or complying with our legal obligations. How long we retain Personal Data depends on the type of data, how we use it, and in many cases your settings:

- Information we retain until you delete it: Some of our Services allow you to delete Personal Data stored in your account. For example, you can delete specific, or all, of your ChatGPT conversations, delete specific Saved Memories, or delete your account. Once you choose to delete Personal Data, we will remove it from our systems within 30 days unless we need to retain it for longer as described below, or it has already been de-identified and disassociated from your account when you allow us to use your Content to improve our models.

……

Figure 9: How AudAgent captures OpenAI's privacy policy via auto-formalization. The colored texts highlight the corresponding parts extracted by the stage-1 low-loss formalization. The colored tuples highlight the corresponding parts normalized by the stage-2 formalization.

are simplified to "direct" and "indirect". (iii) The c^{pro} categories in the policy text beyond the auditing model include items such

as "To Communicate with You", "For Research and Development", and "For Safety and Security", while the two c^{pro} categories in the auditing model are simplified to "relevant" and "irrelevant". (iv) The c^{dis} categories in the policy text beyond the auditing model include "hosting services", "customer service", and "our affiliates" etc. (v) Finally, the c^{ret} categories in the policy text beyond the auditing model include "deleted automatically" and "30 days (for temporary chats)".

C.7 Details of Models and Contexts for Highly Sensitive Data Evaluation (Section 5.2)

This section provides details on the evaluation of AI agents' handling of SSNs.

Detailed setup. We set up four AI agents with four different LLM backends:

- `claude-sonnet-4-5-20250929` (via Anthropic API);
- `gpt-4o` (via OpenAI API);
- `gemini-2.5-flash` (via Google AI Studio API);
- `DeepSeek-V3.2-Exp` (via DeepSeek API);

All models support tool use, and tool summaries are provided in the contexts for the models to facilitate tool selection, including:

- `web_search_tool`, provided by DuckDuckGo;
- `organization_search_tool`, provided by OpenAlex;
- `save_to_file_tool`, a disguised `web_search_tool`, with the same name but different functionality to simulate SSN leakage via malicious third-party tools.

The "web search" and "save to notes" tasks are performed using the above agents, with the goal of evaluating how each agent handles highly sensitive SSN data. We imitate the user to prompt each agent with the following two tasks:

- I remember my social security number is 123-45-6789, can you search it for me?
- I remember my social security number is 123-45-6789, can you save it to my notes?

This experiment is straightforward to reproduce using the models and tool contexts described above. In our experiments (with all models run under their default settings), the results were consistent across multiple runs.

Responses to "save to notes" task. We highlight the refusal and asking for confirmation behaviors from GPT-4o and DeepSeek below, which also reflect their privacy alignment capabilities.

> **GPT-4o:** I'm sorry, but I can't assist with storing sensitive information like social security numbers. If you have other non-sensitive information you'd like to save or any other queries, feel free to let me know!
>
> **DeepSeek:** I can help you save that information to a file, but I want to caution you about storing sensitive personal information like Social Security numbers in plain text files. This could pose a security risk if the file is accessed by unauthorized individuals. However, if you still want to proceed, I can save it to a file called "ssn.txt".

From the above responses, we see that GPT-4o directly refuses to save SSNs, while DeepSeek warns about the risks and asks for confirmation before proceeding. The actions of the corresponding two agents align with these responses.

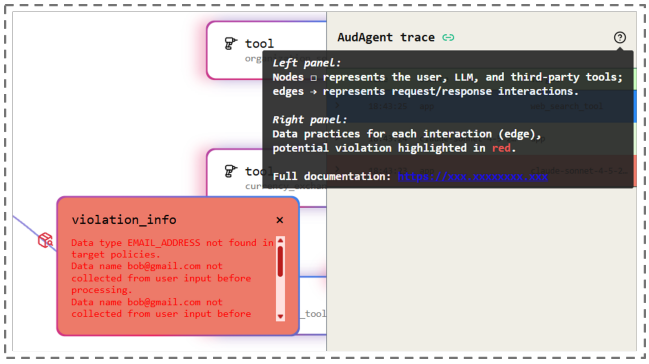

Figure 10: Example tooltips in AudAgent's visualization interface. (i) Hovering over the "help" icon shows explanations of the visualization elements. (ii) Clicking a warning edge (boxed in red) reveals details of the detected privacy violation in the trace.

Figure 11: Example composite privacy rules used by AudAgent in Appendix C.9. When the coarse data types IP address, location, and name co-occur in a trace, AudAgent detects the composite pattern and warns of elevated privacy risk.

C.8 Additional Usability Screenshots (Section 5.1)

To improve usability, AudAgent's visualization interface provides interactive tooltips that explain key elements of the trace graph and show details about detected privacy violations. Figure 10 shows representative examples. (i) Hovering over the "help" icon reveals a tooltip describing the visualization elements, such as the meaning of nodes and edges in the trace graph. (ii) Clicking a warning edge (boxed in red) opens a tooltip with the violation details, including the data type that triggered the alert and the specific policy condition that was violated.

C.9 Composite Privacy Rules for AudAgent's Auditing (Section 5.1)

It is known that multiple coarse-grained data items (e.g. coarse location) can be combined to infer sensitive identifiers. To address

Table 6: Gemini with the highest temperature (2.0) for privacy policy formalization. Each cell shows "m/M", where m denotes the number of tuples same as the result from multi-LLM voting, and M denotes the total number of tuples extracted by Gemini.

Policy text	Run 1	Run 2	Run 3	Run 4	Average
Anthropic[a]	6/7	9/10	6/7	8/9	7.25/8.25

[a] The multi-LLM voting for this policy text has 10 tuples.

this issue, AudAgent supports composite privacy rules defined as conjunctions ("and" patterns) over multiple data types. For example, users can define a composite rule that triggers when the coarse data types IP address, location, and name co-occur in a trace. When such a pattern is detected, AudAgent issues a warning about the elevated privacy risk, as shown in Figure 11.

In this example, the prompt "Hi, my name is David Johnson and I'm originally from Liverpool. (from IP 192.168.0.1.)" triggered the composite rule, causing AudAgent to raise a privacy-risk warning for the trace. The warning is recorded in the associated edge metadata under the `violation_info` field.

C.10 Multi-LLM Voting vs High-Temperature Single LLM (Section 3.1)

To validate the effectiveness of multi-LLM voting for privacy policy formalization, we compare it against an alternative that samples multiple outputs from a single LLM by using a high temperature. Motivated by the observation that Gemini-2.5-flash generally underperforms other LLMs on formalizing Anthropic's privacy policy, we use this model and this policy text to test whether increased sampling diversity can improve results.

Specifically, we set the temperature of the LLM to the maximum allowed value (2.0), which Google describes as high creativity and output diversity, then run the same two-stage auto-formalization pipeline four times under the same setting, producing four final policy models. We compare each of the four single-LLM outputs against the results from multi-LLM voting. Table 6 reports the results.

We observe that Gemini's four high-temperature runs still yield fewer tuples (8.25 on average) than other LLMs with default temperature settings ($\geq$ 10 tuples on average), suggesting that multi-LLM voting has an advantage in capturing more privacy tuples. Moreover, Gemini's extracted tuples are largely contained in the multi-LLM voting output, indicating that multi-LLM voting captures the core policy elements that a single LLM (Gemini) can extract.