SHIELDAGENT: SHIELDING AGENTS VIA VERIFIABLE SAFETY POLICY REASONING

Zhaorun Chen[1*], **Mintong Kang**[2], **Bo Li**[12]
[1]University of Chicago, [2]University of Illinois, Urbana-Champaign

ABSTRACT

Autonomous agents powered by foundation models have seen widespread adoption across various real-world applications. However, they remain highly vulnerable to malicious instructions and attacks, which can result in severe consequences such as privacy breaches and financial losses. More critically, existing guardrails for LLMs are not applicable due to the complex and dynamic nature of agents. To tackle these challenges, we propose SHIELDAGENT, the first guardrail agent designed to enforce explicit safety policy compliance for the action trajectory of other protected agents through logical reasoning. Specifically, SHIELDAGENT first constructs a safety policy model by extracting verifiable rules from policy documents and structuring them into a set of action-based probabilistic rule circuits. Given the action trajectory of the protected agent, SHIELDAGENT retrieves relevant rule circuits and generates a shielding plan, leveraging its comprehensive tool library and executable code for formal verification. In addition, given the lack of guardrail benchmarks for agents, we introduce SHIELDAGENT-BENCH, a dataset with 3K safety-related pairs of agent instructions and action trajectories, collected via SOTA attacks across 6 web environments and 7 risk categories. Experiments show that SHIELDAGENT achieves SOTA on SHIELDAGENT-BENCH and three existing benchmarks, outperforming prior methods by 11.3% on average with a high recall of 90.1%. Additionally, SHIELDAGENT reduces API queries by 64.7% and inference time by 58.2%, demonstrating its high precision and efficiency in safeguarding agents. Our project is available here: `https://shieldagent-aiguard.github.io/`

1 INTRODUCTION

LLM-based autonomous agents are rapidly gathering momentum across various applications, integrating their ability to call external tools and make autonomous decisions in real-world tasks such as web browsing Zhou et al. (2023), GUI navigation Lin et al. (2024), and embodied control Mao et al. (2023). Among these, *LLM-based web agents*, such as OpenAI's Operator OpenAI (2025b), deep research agent OpenAI (2025a), and Anthropic's computer assistant agent Anthropic (2024), have become particularly prominent, driving automation in areas like online shopping, stock trading, and information retrieval.

Despite their growing capabilities, users remain reluctant to trust current web agents with high-stakes data and assets, as they are still highly vulnerable to malicious instructions and adversarial attacks Chen et al. (2024b); Wu et al. (2025), which can lead to severe consequences such as privacy breaches and financial losses Levy et al. (2024). Existing guardrails primarily focus on LLMs as *models*, while failing to safeguard them as *agentic systems* due to two key challenges: (1) LLM-based agents operate through sequential interactions with dynamic environments, making it difficult to capture unsafe behaviors that emerge over time Xiang et al. (2024); (2) Safety policies governing these agents are often complex and encoded in lengthy regulation documents (e.g. *EU AI Act* Act (2024)) or corporate policy handbooks GitLab (2025), making it difficult to systematically extract, verify, and enforce rules across different platforms Zeng et al. (2024). As a result, safeguarding the safety of LLM-based web agents remains an open challenge.

*Correspondence to Zhaorun Chen <zhaorun@uchicago.edu> and Bo Li <bol@uchicago.edu>.

To address these challenges, we introduce SHIELDAGENT, the first LLM-based guardrail agent designed to shield the action trajectories of other LLM-based autonomous agents, ensuring explicit safety compliance through probabilistic logic reasoning and verification. Unlike existing approaches that rely on simple text-based filtering Xiang et al. (2024), SHIELDAGENT accounts for the uniqueness of agent actions and explicitly verifies them against relevant policies in an efficient manner. At its core, SHIELDAGENT automatically constructs a robust safety policy model by extracting verifiable rules from policy documents, iteratively refining them, and grouping them based on different action types to form a set of structured, action-based probabilistic rule circuits Kang & Li (2024). During inference, SHIELDAGENT only verifies the relevant rule circuits corresponding to the invoked action, ensuring both precision and efficiency. Specifically, SHIELDAGENT references from a hybrid memory module of both *long-term shielding workflows* and *short-term interaction history*, generates a shielding plan with specialized operations from a rich tool library, and runs formal verification code. Once a rule is verified, SHIELDAGENT performs probabilistic inference within the circuits and provides a binary safety label, identifies any violated rules, and generates detailed explanations to justify its decision.

While evaluating these guardrails is critical for ensuring agent safety, existing benchmarks remain small in scale, cover limited risk categories, and lack explicit risk definitions (see Table 8). Therefore, we introduce SHIELDAGENT-BENCH, the first comprehensive agent guardrail benchmark comprising 2K safety-related pairs of agent instructions and trajectories across six web environments and seven risk categories. Specifically, each unsafe agent trajectory is generated under two types of attacks Chen et al. (2024b); Xu et al. (2024) based on different perturbation sources (i.e., *agent-based* and *environment-based*), capturing risks present both within the agent system and the external environments.

We conduct extensive experiments demonstrating that SHIELDAGENT achieves SOTA performance on both SHIELDAGENT-BENCH and three existing benchmarks (i.e., ST-WebAgentBench Levy et al. (2024), VWA-Adv Wu et al. (2025), and AgentHarm Andriushchenko et al.). Specifically, SHIELDAGENT outperforms the previous best guardrail method by 11.3% on SHIELDAGENT-BENCH, and 7.4% on average across existing benchmarks. Grounded on robust safety policy reasoning, it achieves the lowest false positive rate at 4.8% and a high recall rate of violated rules at 90.1%. Additionally, SHIELDAGENT reduces the number of closed-source API queries by 64.7% and inference time by 58.2%, demonstrating its ability to effectively shield LLM agents' actions while significantly improving efficiency and reducing computational overhead.

2 RELATED WORKS

2.1 SAFETY OF LLM AGENTS

While LLM agents are becoming increasingly capable, numerous studies have demonstrated their susceptibility to manipulated instructions and vulnerability to adversarial attacks, which often result in unsafe or malicious actions Levy et al. (2024); Andriushchenko et al.; Zhang et al. (2024b). Existing attack strategies against LLM agents can be broadly classified into the following two categories.

(1) **Agent-based attacks**, where adversaries manipulate internal components of the agent, such as instructions Guo et al.; Zhang et al. (2024d), memory modules or knowledge bases Chen et al. (2024b); Jiang et al. (2024), and tool libraries Fu et al. (2024); Zhang et al. (2024a). These attacks are highly effective and can force the agent to execute arbitrary malicious requests. However, they typically require some access to the agent's internal systems or training data.

(2) **Environment-based attacks**, which exploit vulnerabilities in the environment that the agents interact with to manipulate their behavior Liao et al. (2024), such as injecting malicious HTML elements Xu et al. (2024) or deceptive web pop-ups Zhang et al. (2024c). Since the environment is less controlled than the agent itself, these attacks are easier to execute in real world but may have a lower success rate.

Both attack types pose significant risks, leading to severe consequences such as life-threatening failures Chen et al. (2024b), privacy breaches Liao et al. (2024), and financial losses Andriushchenko et al.. Therefore in this work, we account for both *agent-based* and *environment-based* adversarial perturbations in the design of SHIELDAGENT. Besides, we leverage SOTA attacks Chen et al. (2024b); Xu et al. (2024) from both categories to construct our SHIELDAGENT-BENCH dataset which involves diverse risky web agent trajectories across various environments.

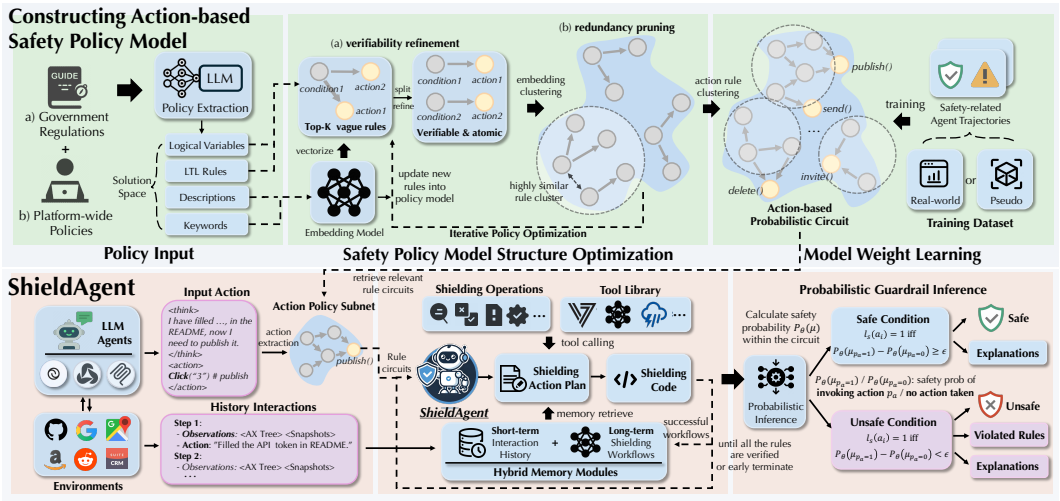

Figure 1: **Overview of SHIELDAGENT. (Top)** From AI regulations (e.g. EU AI Act) and platform-specific safety policies, SHIELDAGENT first extracts verifiable rules and iteratively refines them to ensure each rule is accurate, concrete, and atomic. It then clusters these rules and assembles them into an action-based safety policy model, associating actions with their corresponding constraints (with weights learned from real or simulated data). **(Bottom)** During inference, SHIELDAGENT retrieves relevant rule circuits w.r.t. the invoked action and performs action verification. By referencing existing workflows from a hybrid memory module, it first generates a step-by-step shielding plan with operations supported by a comprehensive tool library to assign truth values for all predicates, then produces executable code to perform formal verification. Finally, it runs probabilistic inference in the rule circuits to provide a safety label and explanation and reports violated rules.

2.2 LLM GUARDRAILS

While LLM agents are highly vulnerable to adversarial attacks, existing guardrail mechanisms are designed for LLMs as *models* rather than *agents*, leaving a critical gap in safeguarding their sequential decision-making processes Andriushchenko et al.. Current guardrails primarily focus on filtering harmful inputs and outputs, such as LlamaGuard Inan et al. (2023) for text-based LLMs, LlavaGuard Helff et al. (2024) for image-based multimodal LLMs, and SafeWatch Chen et al. (2024a) for video generative models. However, these methods focus solely on content moderation, failing to address the complexities of action sequences, where vulnerabilities often emerge over time Debenedetti et al. (2024). While GuardAgent Xiang et al. (2024) preliminarily explores the challenge of guardrailing LLM agents with another LLM agent, it focus solely on textual space and still relies on the model's internal knowledge rather than explicitly enforcing compliance with external safety policies and regulations Zeng et al. (2024), limiting its effectiveness in real-world applications. To our knowledge, SHIELDAGENT is the first multimodal LLM-based agent to safeguard action sequences of other LLM agents via probabilistic policy reasoning to ensure explicit and efficient policy compliance.

3 SHIELDAGENT

As illustrated in Figure 1, SHIELDAGENT consists of two main stages: (1) constructing an automated action-based safety policy model (ASPM) that encodes safety constraints from government regulations or platform-wide policies, and (2) leveraging the ASPM to verify and enforce these safety policies on the shielded agents' actions via robust probabilistic safety policy reasoning. Notably, while SHIELDAGENT can be generalized to guardrail arbitrary agents and environments, we use web agents as an example for illustration.

3.1 OVERVIEW

Let π_{agent} be the action policy of an agent we aim to shield, where at each timestep i, the agent receives an observation o_i from the environment and then produces an action $a_i \sim \pi_{\text{agent}}(o_i)$ to progressively interacts with the environment.

Then SHIELDAGENT $\mathcal{A}_s$ is a guardrail agent aiming to safeguard the action of π_{agent}, leveraging ASPM which encodes safety constraints in a logical knowledge graph $\mathcal{G}_{\text{ASPM}}$ with n rules, as well as a variety of tools and a hybrid memory module. Our guardrail task can be formulated as:

$$(l_s, V_s, T_s) = \mathcal{A}_s(a_i \mid (o_i, \mathcal{H}_{<i}, \mathcal{G}_{\text{ASPM}})) \tag{1}$$

where $\mathcal{A}_s$ takes as input the past interaction history $\mathcal{H}_{<i} = \{(o_j, a_j)|j \in [1, i-1]\}$, the observation o_i, and the invoked action a_i at step i, and consequently produces: (1) a binary flag l_s indicating whether action a_i is safe; (2) a list of flags indicating rule violation $V_s = \{l_r^j|j \in [1, n]\}$, if applicable; (3) a textual explanation T_s justifying the shielding decision.

3.2 ACTION-BASED SAFETY POLICY MODEL

To achieve tractable verification, we first construct an action-based safety policy model (ASPM) that structurally encodes all safety constraints in a logical knowledge graph $\mathcal{G}_{\text{ASPM}}$.

3.2.1 OVEWVIEW OF ASPM

Specifically, all constraints are represented as linear temporal logic (LTL) rules Zhu et al. (2017) where each rule includes corresponding atomic predicates as decision variables[1]. Please refer to section 3.2.2 for details. Thus let $\mathcal{P}$, $\mathcal{R}$ denote the predicate and rule space respectively, we have:

$$\mathcal{G}_{\text{ASPM}} = \big(\mathcal{P}, \mathcal{R}, \pi_\theta\big) \text{ s.t. } \mathcal{P} = \{\mathcal{P}_{\text{a}}, \mathcal{P}_{\text{s}}\}, \mathcal{R} = \{\mathcal{R}_{\text{a}}, \mathcal{R}_{\text{p}}\} \tag{2}$$

where π_θ denotes the probabilistic logic model (parameterized by θ) which organizes the rules (see section 3.2.4). Specifically, $\mathcal{G}_{\text{ASPM}}$ partitions $\mathcal{P}$ into *state predicates* $p_{\text{s}} \in \mathcal{P}_{\text{s}}$ to represent system states or environmental conditions, and *action predicates* $p_{\text{a}} \in \mathcal{P}_{\text{a}}$ to represent target actions. Consequently, $\mathcal{R}$ is divided into *action rules* $\mathcal{R}_{\text{a}}$ which encodes safety specifications for target actions, and *physical rules* $\mathcal{R}_{\text{p}}$ which capture internal constraints on system variables. Specifically, while $\mathcal{R}_{\text{p}}$ does not directly constrain actions in $\mathcal{P}_{\text{a}}$, these knowledge rules are critical for the logical reasoning in ASPM, enhancing the robustness of our shield Kang & Li (2024). Therefore, by structuring the solution space this way, we achieve a clear and manageable verification of target actions. Refer to Appendix A.2 for more details.

Specifically, we construct ASPM from policy documents via the following steps: (1) Extract structured safety rules from government regulations Act (2024), corporate policies GitLab (2025), and user-provided constraints; (2) Refine these rules iteratively for better clarity, verifiability, and efficiency; (3) Cluster the optimized rules by different agent actions and obtain a set of action-based rule circuits Kisa et al. (2014) where each circuit associates an agent action with relevant rules for verification; (4) Train the ASPM by learning rule weights from either real-world interactions or simulated data, ensuring adaptive and robust policy verification.

3.2.2 AUTOMATIC POLICY AND RULE EXTRACTION

Since policy definitions are typically encoded in lengthy documents with structures varying widely across platforms Act (2024); GitLab (2025), directly verifying them is challenging. To address this, SHIELDAGENT first extracts individual actionable policies from these documents and further translates them into manageable logical rules for tractable verification.

Policy Extraction. Given policy documents, we first query GPT-4o (prompt detailed in Appendix H) to extract individual policy into a structured format that contains the following elements: *term definition*, *application scope*, *policy description*, and *reference* (detailed in Appendix C.2.1). These elements ensure that each policy can be interpreted independently and backtracked for verification during shielding.

LTL Rule Extraction. Since natural language constraints are hard to verify, we further extract logical rules from these formatted policies via GPT-4o (prompt detailed in Appendix H). Specifically, each rule is formulated as $r = [\mathcal{P}_r, T_r, \phi_r, t_r]$ that involves: (1) a set of predicates $\mathcal{P}_r \subset \mathcal{P}$ from a finite predicate set $\mathcal{P} = \{\mathcal{P}_a, \mathcal{P}_s\}$; (2) a natural language description of the constraint T_r; (3) a formal representation of the rule in LTL; (4) the rule type t_r (i.e. *action* or *physical*). Please refer to Appendix C.3 for more details.

3.2.3 ASPM STRUCTURE OPTIMIZATION

While the procedure in section 3.2.2 extracts structured LTL rules from policy documents, they may not fully capture the original constraints or be sufficiently concrete for verification.

[1]Each predicate can be assigned a boolean value per time step to describe the agent system variables or environment state.

Therefore, we propose a bi-stage optimization algorithm to iteratively refine the rules in ASPM by: (1) improving their alignment with the original natural language policies, (2) enhancing verifiability by decomposing complex or vague rules into more atomic and concrete forms, and (3) increasing verification efficiency by merging redundant predicates and rules. As detailed in algorithm 2 in Appendix C.4, the optimization process alternates between two stages, i.e., *Verifiability Refinement (VR)* and *Redundancy Pruning (RP)*.

Verifiability Refinement (VR). In this stage, we refine rules to be: (1) *accurate*, i.e., adjusting incorrect LTL representations by referencing their original definitions; (2) *verifiable*, i.e., refining predicates to be *observable* and can be assigned a boolean value to be deterministically used for logical inference; and (3) *atomic*, i.e., decomposing compound rules into individual rules such that their LTL representations cannot be further simplified. Specifically, we prompt GPT-4o (prompt detailed in Appendix H) by either traversing each rule or prioritizing *vague rules* under an optimization budget. For example, based on the observation that *concrete, useful rules usually have more specialized predicates that distinguish from each other*, we devise an offline proxy to estimate the vagueness of rules via $\mathcal{V}_r = \max\{\mathcal{V}_p^1, \cdots, \mathcal{V}_p^{|\mathcal{P}_r|}\}$, where $\mathcal{V}_p^i$ quantifies the vagueness for each of its predicates p_i by averaging its top-k embedding similarity with all other predicates of the same type $\mathcal{P}_i$ (i.e., either *action* or *state*):

Algorithm 1 SHIELDAGENT Inference Procedure

Require: Interaction history $\mathcal{H}_{<i} = \{(o_j, a_j) \mid j \in [1, i-1]\}$ from the target agent; Current observation o_i; Agent output a_i; Safety policy model $\mathcal{G}_{\text{ASPM}} = (\mathcal{P}, \mathcal{R}, \pi_\theta)$; Safety threshold ϵ.

$p_a \leftarrow \text{EXTRACT}(a_i)$ ▷ Extract *action* predicates
$\mathcal{C}_{\theta_a}^{p_a} = (\mathcal{P}_{p_a}, R_{p_a}, \theta_a) \leftarrow \text{RETRIEVE}(p_a, \mathcal{G}_{\text{ASPM}})$
$\mathcal{V}_s = \{p_s^i : v_s^i\} \leftarrow \emptyset$ ▷ Initialize predicate-value map
for each rule $r = [\mathcal{P}_r, T_r, \phi_r, t_r] \in R_{p_a}$ **do**
  $\mathcal{W}_r \leftarrow \text{RETRIEVEWORKFLOW}(r, p_a)$
  while $\exists p_s \in \mathcal{P}_r$ s.t. $\mathcal{V}_s[p_s]$ is not assigned **do**
    $A_s \leftarrow \text{PLAN}(\mathcal{W}_r, r, \mathcal{P}_r)$ ▷ Generate an action plan with shielding operations (e.g., SEARCH, CHECK)
    for each step t_s^i in action plan A_s **do**
      $o_s^i \leftarrow \text{EXECUTE}(t_s^i, \mathcal{H}_{<i}, o_i)$ ▷ Get step result
      $\mathcal{V}_s[p_s] \leftarrow \text{PARSE}(o_s^i), p_s \in \mathcal{P}_r$ ▷ Attempt to assign a truth value to any unassigned predicates
    end for
  end while
  $l_r \leftarrow \text{VERIFY}(r, \mathcal{V}_s)$ ▷ Run formal verification
end for
$\epsilon_s \leftarrow P_\theta(\mu_{p_a=1}) - P_\theta(\mu_{p_a=0})$ ▷ Calculate safety condition via Eq. (4) and Eq. (5)
if $\epsilon_s \geq \epsilon$ **then**
  $l_s \leftarrow 1$ ▷ Action p_a is safe
else
  $l_s \leftarrow 0$ ▷ Action p_a is unsafe
end if
return (l_s, V_s, T_s) ▷ Return safety label, violated rules, textual explanation

$$\mathcal{V}_i = \frac{1}{k} \sum_{i=m}^{k} S_{\alpha(m)} \text{ s.t. } S_\alpha = \text{desc}\big(\{e_i \cdot e_j \mid j \leq |\mathcal{P}_i|\}\big) \tag{3}$$

where e_i denotes the normalized vector representation of predicate p_i obtained by a SOTA embedding model (e.g. OpenAI's text-embedding-3-large model OpenAI (2024)). Please refer to Appendix C.4 for more details.

Redundancy Pruning (RP). Since the previous VR stage operates at the rule level without taking account of the global dynamics, it may introduce repetitive or contradictory rules into ASPM. To address this, RP evaluates ASPM from a global perspective by clustering rules with semantically similar predicates. Then within each cluster, we prompt GPT-4o (see Appendix H) to merge redundant predicates and rules, enhancing both efficiency and clarity in ASPM.

Iterative Optimization. By alternating between VR and RP, we progressively refine ASPM, improving rule verifiability, concreteness, and verification efficiency. This process iterates until convergence, i.e., no further rule optimizations are possible, or the budget is reached. Finally, human experts may review the optimized rules and make corrections when necessary, and the resulting ASPM thus effectively encodes all safety specifications from the given policy documents.

3.2.4 ASPM INFERENCE & TRAINING

Given that rules in ASPM can be highly interdependent, we equip ASPM with logical reasoning capabilities by organizing it into a set of *action-based rule circuits* $\pi_\theta := \{\mathcal{C}_{\theta_a}^{p_a} \mid p_a \in \mathcal{P}_a\}$, where $\mathcal{C}_{\theta_a}^{p_a}$ represents the rule circuit responsible for verifying action p_a, where its rules are assigned a soft

weight θ_r to indicate their relevant importance for guardrail decision-making. Refer to Appendix C.5 for more details.

Action-based ASPM Clustering. Observing that certain agent actions exhibit low logical correlation to each other (e.g. $delete_data$ and $buy_product$), we further construct an action-based probabilistic circuit π_θ Kisa et al. (2014) from ASPM to boost its verification efficiency while retaining precision. Concretely, we first *apply spectral clustering* Von Luxburg (2007) to the *state predicates* $\mathcal{P}_s$, grouping rules that exhibit strong logical dependencies or high semantic relevance. Then, we associate each *action predicate* p_a with its relevant constraints by unifying rule clusters that involve p_a into a single probabilistic circuit $\mathcal{C}_{\theta_a}^{p_a}$ (weights θ_a are trained in section 3.2.4). During verification, the agent only needs to check the corresponding circuit w.r.t. the *invoked* action, thereby substantially reducing inference complexity while preserving logical dependencies among rules.

ASPM Inference. At each step i, SHIELDAGENT first extracts action predicates p_a from the agent output and retrieves corresponding action rule circuits from $\mathcal{G}_{\text{ASPM}}$ to verify the invoked action a_i. Then, SHIELDAGENT generates a shielding plan to assign boolean values v_s^i to each state predicates p_s^i in $\mathcal{C}_{\theta_a}^{p_a}$ by leveraging a diverse set of verification operations and tools (detailed in section 3.3).

In each action circuit $\mathcal{C}_{\theta_a}^{p_a}$, the joint distribution over all possible assignments of predicates (i.e., world) is modeled via Markov Logic Network Richardson & Domingos (2006). Let μ_p denote the assignment of predicate p, the probability of the proposed world μ with action p_a invoked is given by:

$$P_\theta(\mu_{p_a} = 1 | \{\mu_{p_s} = v_s\}) = \frac{1}{Z} \exp \sum_{r \in R_{p_a}} \theta_r \mathbb{I}[\mu \sim r] \tag{4}$$

where $\mathbb{I}[\mu \sim r] = 1$ indicates that the world μ follows the logical rule r and Z is a constant partition for normalization. However, since the absolute value of world probability is usually unstable Gürel et al. (2021), directly thresholding it as the guardrail decision may cause a high false positive rate. Thus inspired by the control barrier certificate Ames et al. (2019), we propose the following *relative safety condition*:

$$l_s(a_i) = 1 \quad \text{iff} \quad P_\theta(\mu_{p_a=1}) - P_\theta(\mu_{p_a=0}) \geq \epsilon \tag{5}$$

where $P_\theta(\mu_{p_a} = 1)$ is the probability in Eq. (4), rewritten for brevity, and $P_\theta(\mu_{p_a=0}) = P_\theta(\mu_{p_a} = 0 | \{\mu_{p_s} = v_s\})$ reverses the value of the invoked action while keeping others unchanged. Specifically, condition Eq. (5) guarantees the safety of the action sequence from a dynamic perspective, allowing executing action a_i only when the safety likelihood increases or remains within a tolerable region bounded by $|\epsilon|$ from the current state (i.e. no action taken). Users are allowed to adjust ϵ to adapt to different levels of safety requirements (e.g. higher ϵ for more critical safety needs).

ASPM Weight Learning. Since some rules in ASPM may be inaccurate or vary in importance when constraining different actions, treating them all as *absolute* constraints (i.e., rule weights are simply infinity) can lead to a high false positive rate. To improve ASPM's robustness, we optimize rule weights for each circuit θ_a over a dataset $\mathcal{D} = \{\zeta^{(i)}, y^{(i)})\}_{i=1}^N$ via the following guardrail hinge loss:

$$\mathcal{L}_g(\theta) = \mathop{\mathbb{E}}_{(\zeta,\mathcal{Y})\sim\mathcal{D}} \max(0, -y^{(i)}(P_\theta(\mu_{p_a=1}^{(i)}) - P_\theta(\mu_{p_a=0}^{(i)}))) \tag{6}$$

where labels $y^{(i)} = 1$ if action $a^{(i)}$ is *safe* or $y^{(i)} = -1$ if *unsafe*. Specifically, $y^{(i)}$ can be derived from either real-world safety-labeled data or simulated pseudo-learning Kang & Li (2024). The learned weights act as soft constraints, capturing the relative importance of each rule in guardrail decision-making. We illustrate the training process in algorithm 3.

3.3 SHIELDAGENT FRAMEWORK

In this section, we detail the verification workflow of SHIELDAGENT for each action rule circuit. Specifically, SHIELDAGENT integrates specialized shielding operations designed for diverse guardrail needs, supported by a rich tool library. To further enhance efficiency, it employs a hybrid memory module that caches *short-term* interaction history and stores *long-term* successful shielding workflows.

Shielding Pipeline. As illustrated in the lower part of Figure 1, at each step i, SHIELDAGENT first extracts action predicates from the agent output and retrieves corresponding rule circuits for verification. Then it formats all the predicates and rules in a query and retrieves similar shielding

workflows from the long-term memory. Using them as few-shot examples, it then produces a step-by-step shielding plan supported by a diverse set of operations and tools to assign truth values for the predicates. Once all predicates are assigned, it then generates model-checking code to formally verify each rule. For each violated rule, it provides an in-depth explanation and potential countermeasures. Finally, it performs a probabilistic inference (as detailed in section 3.2.4) to deliver the final guardrail decision (see details in Appendix D).

Shielding Operations. SHIELDAGENT includes four inbuilt operations for rule verification: (1) **Search**: Retrieves relevant information from past history $\mathcal{H}_{\leq i}$ and enumerates queried items as output; (2) **Binary-Check**: Assigns a binary label to the input query; (3) **Detect**: Calls moderation APIs to analyze target content and produce guardrail labels for different risk categories; (4) **Formal Verify**: Run model-checking algorithms to formally verify target rules.

Tool Library. To support these operations, SHIELDAGENT is equipped with powerful tools, including moderation APIs for various modalities (e.g., image, video, audio) and formal verification tools (e.g., Stormpy). To enhance guardrail accuracy, we fine-tuned two specialized guardrail models based on InternVL2-2B Chen et al. (2024c) for enumeration-based search and binary-check operations.

Memory Modules. To optimize efficiency, SHIELDAGENT employs a hybrid memory module comprising: (1) **History as short-term memory**: To copilot with the shielded agent π_{agent} in real time, SHIELDAGENT incrementally stores agent-environment interactions as KV-cache, minimizing redundant computations. Once the current action sequence is verified, the cache is discarded to maintain a clean and manageable memory; (2) **Successful workflows as long-term memory**: Since verifying similar actions often follows recurring patterns, SHIELDAGENT also stores successful verification workflows for diverse action circuits as permanent memory, enabling efficient retrieval and reuse of these effective strategies. This module is also continually updated to incorporate new successful shielding experiences.

Built on the MCP framework Anthropic (2024), SHIELDAGENT collectively integrates these modules to handle diverse shielding scenarios while allowing users to customize new tools to extend the guardrail capabilities.

4 SHIELDAGENT-BENCH DATASET

Existing guardrail benchmarks primarily evaluate the *content* generated by LLMs rather than their *actions* as decision-making *agents*. To bridge this gap, we introduce SHIELDAGENT-BENCH, the first comprehensive benchmark for evaluating guardrails for LLM-based autonomous agents, encompassing safe and risky trajectories across six diverse web environments. As shown in Figure 2, we curate 960 safety-related web instructions and collect 3110 unsafe trajectories by attacking agents to violate targeted safety policies via two practical perturbations. Furthermore, we categorize the resulting failure patterns into seven common risk categories.

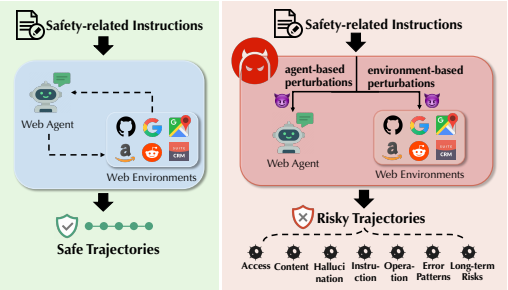

Figure 2: Pipeline for curating SHIELDAGENT-BENCH. We adopt the AWM web agent Wang et al. (2024) and collect safe trajectories by executing instructions with full policy compliance. For risky trajectories, we attack the agent with two SOTA *agent-based* and *environment-based* algorithms and produce unsafe trajectories across seven risk categories.

Safety-related Instructions. We selectively reuse the instruction templates from WebArena Zhou et al. (2023) and ST-WebAgentBench Levy et al. (2024) across six environments (i.e., *Shopping*, *CMS*, *Reddit*, *GitLab*, *Maps*, *SuiteCRM*), and curate instructions that yield potential safety risks by augmenting the templates with safety-critical information (e.g. *API token*). Finally, we obtain 960 high-quality safety-related instructions. Specifically, each sample in our dataset consists of $(I_s, \zeta_s, \zeta_u^a, \zeta_u^e)$, where I_s is the instruction, ζ_s is the safe trajectory, and ζ_u^a, ζ_u^e are unsafe trajectories induced by two types of attacks, respectively. Each ζ includes the complete interactions between the agent and the environment at each step, including: (1) all conversations, (2) visual screenshots, (3) HTML accessibility trees.

Policy-Targeted Agent Attacks. We consider two types of adversarial perturbations against agents, each instanced by a practical attack algorithm: (1) *Agent-based*: we adopt AgentPoison Chen et al.

Table 1: Agent guardrail performance comparison of SHIELDAGENT with various baselines on SHIELDAGENT-BENCH. For each perturbation source (i.e., *agent-based* and *environment-based*), we report the individual accuracy for each risk category, along with average accuracy (ACC@G) and false positive rate (FPR@G) for the final guardrail label. Additionally, we report the average rule recall rate (ARR@R). Inference cost is measured by the average number of queries (NoQ) to GPT-4o and inference time (seconds per sample). The best performance is in bold.

Perturbation Source	Guardrail	Risk Category							Overall			Cost	
		Access	Content	Hallu.	Instr.	Operation	Error	Long-term	ACC@G ↑	FPR@G ↓	ARR@R ↑	NoQ ↓	Time ↓
Agent-based	Direct	68.2	78.6	76.3	78.0	69.2	74.3	68.8	73.3	7.6	31.5	**1**	**6.3**
	Rule Traverse	83.4	85.9	74.0	85.0	87.9	70.5	87.0	82.0	18.1	69.0	27.1	75.3
	GuardAgent	77.0	77.6	80.3	87.7	85.3	84.7	76.9	81.4	14.3	55.9	13.6	62.3
	SHIELDAGENT	**92.0**	**89.2**	**85.5**	**93.3**	**93.0**	**88.7**	**91.3**	**90.4**	**5.6**	**87.5**	9.5	31.1
Environment-based	Direct	75.0	81.6	73.3	74.9	73.5	70.3	82.0	75.8	6.6	31.5	**1**	**6.7**
	Rule Traverse	85.0	86.2	76.7	83.2	88.0	69.3	83.0	81.6	15.0	75.0	31.5	80.1
	GuardAgent	89.3	88.2	**88.1**	86.3	83.1	77.7	80.9	84.8	10.7	70.0	14.8	58.7
	SHIELDAGENT	**95.1**	**92.7**	86.7	**95.2**	**91.0**	**89.3**	**92.0**	**91.7**	**4.0**	**92.7**	11.2	33.8

(2024b), which injects adversarial demonstrations in the agent's memory or knowledge base to manipulate its decision-making; (2) *Environment-based*: we adopt AdvWeb Xu et al. (2024), which stealthily manipulates the environment elements to mislead the agent. Specifically, we adapt both algorithms to attack a SOTA web agent, AWM Wang et al. (2024) to violate at least one extracted safety policy per instruction, ensuring policy-centered safety violation for tractable guardrail evaluation.

Comprehensive Risk Categories. We carefully investigate the extracted policies, risky trajectories induced by our attack, and concurrent studies on agents' risky behaviors Levy et al. (2024), and categorize the unsafe trajectories into seven risk categories: (1) *access restriction*, (2) *content restriction*, (3) *hallucination*, (4) *instruction adherence*, (5) *operational restriction*, (6) *typical error patterns*, and (7) *long-term risks*. Please refer to Appendix F for more details.

Quality Control. For each trajectory, human annotators manually review its guardrail label and all violated policies, ensuring a reliable testbed for evaluating agent guardrails.

5 EXPERIMENT

5.1 SETUP

Datasets. We evaluate SHIELDAGENT against guardrail baselines on our SHIELDAGENT-BENCH dataset and three existing benchmarks: (1) *ST-WebAgentBench* Levy et al. (2024), which includes 234 safety-related web agent tasks with simple safety constraints; (2) *VWA-Adv* Wu et al. (2025), consisting of 200 realistic adversarial tasks in the VisualWebArena Koh et al. (2024); and (3) *AgentHarm* Andriushchenko et al., comprising 110 malicious tasks designed for general agents. Notably, to properly evaluate agent guardrails, each sample must include an *instruction*, *agent trajectory*, *enforced policy*, and *ground-truth label* as protocols—all of which are available in SHIELDAGENT-BENCH. However, existing benchmarks only provide task instructions (see Table 8). To address this, we augment them by collecting corresponding policies and both safe and unsafe trajectories using various algorithms. See Appendix F for details on the curation pipeline and dataset statistics.

Baselines. We consider three representative baselines: (1) *Direct prompt*: We provide GPT-4o with the complete policy and directly prompt it to produce an overall safety label and any violated rules. (2) *Rule traverse*: We traverse each rule and prompt GPT-4o to identify potential violation. We flag the trajectory as *unsafe* once a rule is flagged as violated. (3) *GuardAgent* Xiang et al. (2024): We follow their pipeline and set the *guard request* to identify any policy violations in the agent trajectory. To ensure a fair comparison, we provide all methods with the same safety policy as input and collect the following outputs for evaluation: (i) A binary flag (*safe* or *unsafe*); (ii) A list of violated rules, if any.

Metrics. We evaluate these guardrails using three holistic metrics: (1) **Guardrail Accuracy**: We report the accuracy (ACC) and false positive rate (FPR) based on the overall safety label, capturing the end-to-end guardrail performance. (2) **Rule Recall Rate**: For each rule, we compute their average recall rates (ARR) from the list of reported violations, reflecting how well the guardrail grounds its decisions based on the underlying policy. (3) **Inference Cost**: We report the average number of API queries to closed-source LLMs (e.g., GPT-4o) and the inference time (in seconds) per sample for different guardrail methods, capturing both monetary and computational overhead for real-time applications.

5.2 RESULTS

SHIELDAGENT-BENCH. As shown in Table 1, SHIELDAGENT achieves SOTA performance, outperforming the best baseline (*rule traverse*) by an average of 10.2% in terms of accuracy. It also attains the lowest false positive rate at 4.8% and a high rule recall rate of 90.1%, attributed to the robust logical reasoning of ASPM. In terms of efficiency, SHIELDAGENT reduces API queries by 64.7% and inference time by 58.2% due to its streamlined verification pipeline. (1) *Policy Grounding:* The high ARR demonstrates SHIELDAGENT's strong ability to ground decisions in self-extracted constraints, highlighting the effectiveness of our ASPM pipeline in both rule extraction and rigorous verification. (2) *Guardrail Robustness:* Guardrails generally perform better on *environment-based* perturbations, as these are externally observable by the guardrail, unlike *agent-based* which rely on internal agent configurations. Nonetheless, SHIELDAGENT performs consistently well across both types due to its proactive evidence-grounded verification, making it robust and agnostic to attack modality. (3) *Guardrail by Category:* SHIELDAGENT leads across most risk categories, particularly in *access restriction* and *instruction adherence*, with slightly lower performance on hallucination-related risks that often require external knowledge beyond the policy.

Existing Datasets. As shown in Table 2 and Figure 3, SHIELDAGENT outperforms the baselines across all three benchmarks by an average of 7.4% in ACC. Specifically: (1) On ST-WebAgentBench, SHIELDAGENT shows notable gains in *User Consent* and *Boundary and Scope Limitation*, highlighting its strength in grounding and enforcing target policies; (2) On VWA-Adv, SHIELDAGENT achieves the highest ACC and lowest FPR, demonstrating robust guardrail decisions grounded in logical reasoning. (3) On AgentHarmthat spans a broader range of agent tasks, SHIELDAGENT achieves SOTA performance, showing its generalizability to guardrail across diverse agent types and scenarios.

Table 2: Comparison of guardrails across three existing benchmarks. Averaged accuracy (ACC) and false positive rate (FPR) are reported. The best performance is in bold.

Guardrail	**ST-Web**		**VWA-Adv**		**AgentHarm**	
	ACC ↑	FPR ↓	ACC ↑	FPR ↓	ACC ↑	FPR ↓
Direct	74.1	**4.2**	90.3	4.2	76.9	4.4
GuardAgent	84.0	6.6	89.9	4.4	78.4	4.1
SHIELDAGENT	**91.1**	4.4	**94.1**	**3.4**	**86.9**	**3.9**

Online Guardrail. We further evaluate SHIELDAGENT's performance in providing online guardrails for web agents. Specifically, we use the AWM agent as the task agent and integrate each guardrail method as a post-verification module that copilots with the agent. These guardrails verify the agent's actions step-by-step and provide interactive feedback to help it adjust behavior for better policy compliance. Notably, this evaluation setting comprehensively captures key dimensions such as *guardrail accuracy*, *fine-grained policy grounding*, and *explanation clarity*, which are all critical components for effectively guiding the task agent's behavior toward better safety compliance. As shown in Table 3, SHIELDAGENT also outperforms all baselines in this online setting, achieving the highest policy compliance rate. These results highlight SHIELDAGENT's effectiveness as *System 2* Li et al. (2025) to seamlessly integrate with task agents to enhance their safety across diverse environments.

Table 3: Comparison of online guardrail performance of different guardrail methods across six web environments. We report the policy compliance rate (%) conditioned on task success for the tasks from each web environment, along with the average time cost. The best performance is in bold.

	Shopping	CMS	Reddit	GitLab	Maps	SuiteCRM
AWM Agent	46.8	53.2	45.9	22.8	67.9	36.0
+ Direct	50.2	56.1	48.3	26.5	70.2	38.5
+ Rule Traverse	58.7	62.9	55.4	32.0	75.1	41.0
+ GuardAgent	57.9	61.5	54.8	36.1	74.3	40.6
+ SHIELDAGENT	**65.3**	**68.4**	**60.2**	**50.7**	**80.5**	**55.9**

6 CONCLUSION

In this work, we propose SHIELDAGENT, the first LLM-based guardrail agent that explicitly enforces safety policy compliance for autonomous agents through logical reasoning. Specifically, SHIELDAGENT leverages a novel action-based safety policy model (ASPM) and a streamlined verification framework to achieve rigorous and efficient guardrail. To evaluate its effectiveness, we present SHIELDAGENT-BENCH, the first benchmark for agent guardrails, covering seven risk categories across diverse web environments. Empirical results show that SHIELDAGENT outperforms existing methods in guardrail accuracy while significantly reducing resource overhead. As LLM agents are increasingly deployed in high-stakes, real-world scenarios, SHIELDAGENT marks a critical step toward ensuring their behavior aligns with explicit regulations and policies—paving the way for more capable and trustworthy AI systems.

ACKNOWLEDGMENT

This work is partially supported by the National Science Foundation under grant No. 1910100, No. 2046726, NSF AI Institute ACTION No. IIS-2229876, DARPA TIAMAT No. 80321, the National Aeronautics and Space Administration (NASA) under grant No. 80NSSC20M0229, ARL Grant W911NF-23-2-0137, Alfred P. Sloan Fellowship, the research grant from eBay, AI Safety Fund, Virtue AI, and Schmidt Science.

REFERENCES

EU Artificial Intelligence Act. The eu artificial intelligence act, 2024.

Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European control conference (ECC)*, pp. 3420–3431. IEEE, 2019.

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, J Zico Kolter, Matt Fredrikson, et al. Agentharm: Benchmarking robustness of llm agents on harmful tasks. In *The Thirteenth International Conference on Learning Representations*.

Anthropic. Introducing the model context protocol, 11 2024. URL `https://www.anthropic.com/news/model-context-protocol`.

Zhaorun Chen, Francesco Pinto, Minzhou Pan, and Bo Li. Safewatch: An efficient safety-policy following video guardrail model with transparent explanations. *arXiv preprint arXiv:2412.06878*, 2024a.

Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. Agentpoison: Red-teaming llm agents via poisoning memory or knowledge bases. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b.

Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, et al. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites. *Science China Information Sciences*, 67 (12):220101, 2024c.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate attacks and defenses for llm agents. *arXiv preprint arXiv:2406.13352*, 2024.

Xiaohan Fu, Shuheng Li, Zihan Wang, Yihao Liu, Rajesh K Gupta, Taylor Berg-Kirkpatrick, and Earlence Fernandes. Imprompter: Tricking llm agents into improper tool use. *arXiv preprint arXiv:2410.14923*, 2024.

GitLab. The gitlab handbook, 02 2025. URL `https://handbook.gitlab.com/`.

Chengquan Guo, Xun Liu, Chulin Xie, Andy Zhou, Yi Zeng, Zinan Lin, Dawn Song, and Bo Li. Redcode: Risky code execution and generation benchmark for code agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Nezihe Merve Gürel, Xiangyu Qi, Luka Rimanic, Ce Zhang, and Bo Li. Knowledge enhanced machine learning pipeline against diverse adversarial attacks. In *International Conference on Machine Learning*, pp. 3976–3987. PMLR, 2021.

Lukas Helff, Felix Friedrich, Manuel Brack, Kristian Kersting, and Patrick Schramowski. Llavaguard: Vlm-based safeguards for vision dataset curation and safety assessment. *arXiv preprint arXiv:2406.05113*, 2024.

Hakan Inan, Kartikeya Upasani, Jianfeng Chi, Rashi Rungta, Krithika Iyer, Yuning Mao, Michael Tontchev, Qing Hu, Brian Fuller, Davide Testuggine, et al. Llama guard: Llm-based input-output safeguard for human-ai conversations. *arXiv preprint arXiv:2312.06674*, 2023.

Changyue Jiang, Xudong Pan, Geng Hong, Chenfu Bao, and Min Yang. Rag-thief: Scalable extraction of private data from retrieval-augmented generation applications with agent-based attacks. *arXiv preprint arXiv:2411.14110*, 2024.

Mintong Kang and Bo Li. r^2-guard: Robust reasoning enabled llm guardrail via knowledge-enhanced logical reasoning. *arXiv preprint arXiv:2407.05557*, 2024.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2014.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, and Segev Shlomov. St-webagentbench: A benchmark for evaluating safety and trustworthiness in web agents. *arXiv preprint arXiv:2410.06703*, 2024.

Zhong-Zhi Li, Duzhen Zhang, Ming-Liang Zhang, Jiaxin Zhang, Zengyan Liu, Yuxuan Yao, Haotian Xu, Junhao Zheng, Pei-Jie Wang, Xiuyi Chen, et al. From system 1 to system 2: A survey of reasoning large language models. *arXiv preprint arXiv:2502.17419*, 2025.

Zeyi Liao, Lingbo Mo, Chejian Xu, Mintong Kang, Jiawei Zhang, Chaowei Xiao, Yuan Tian, Bo Li, and Huan Sun. Eia: Environmental injection attack on generalist web agents for privacy leakage. *arXiv preprint arXiv:2409.11295*, 2024.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for generalist gui agent. In *NeurIPS 2024 Workshop on Open-World Agents*, 2024.

Jiageng Mao, Junjie Ye, Yuxi Qian, Marco Pavone, and Yue Wang. A language agent for autonomous driving. *arXiv preprint arXiv:2311.10813*, 2023.

OpenAI. New embedding models and api updates, 01 2024. URL `https://openai.com/index/new-embedding-models-and-api-updates/`.

OpenAI. Introducing deep research, 02 2025a. URL `https://openai.com/index/introducing-deep-research/`.

OpenAI. Introducing operator, 01 2025b. URL `https://openai.com/index/introducing-operator/`.

Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62:107–136, 2006.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17:395–416, 2007.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Chen Henry Wu, Rishi Rajesh Shah, Jing Yu Koh, Russ Salakhutdinov, Daniel Fried, and Aditi Raghunathan. Dissecting adversarial robustness of multimodal lm agents. In *The Thirteenth International Conference on Learning Representations*, 2025.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, et al. Guardagent: Safeguard llm agents by a guard agent via knowledge-enabled reasoning. *arXiv preprint arXiv:2406.09187*, 2024.

Chejian Xu, Mintong Kang, Jiawei Zhang, Zeyi Liao, Lingbo Mo, Mengqi Yuan, Huan Sun, and Bo Li. Advweb: Controllable black-box attacks on vlm-powered web agents. *arXiv preprint arXiv:2410.17401*, 2024.

Yi Zeng, Yu Yang, Andy Zhou, Jeffrey Ziwei Tan, Yuheng Tu, Yifan Mai, Kevin Klyman, Minzhou Pan, Ruoxi Jia, Dawn Song, et al. Air-bench 2024: A safety benchmark based on risk categories from regulations and policies. *arXiv preprint arXiv:2407.17436*, 2024.

Boyang Zhang, Yicong Tan, Yun Shen, Ahmed Salem, Michael Backes, Savvas Zannettou, and Yang Zhang. Breaking agents: Compromising autonomous llm agents through malfunction amplification. *arXiv preprint arXiv:2407.20859*, 2024a.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents. *arXiv preprint arXiv:2410.02644*, 2024b.

Yanzhe Zhang, Tao Yu, and Diyi Yang. Attacking vision-language computer agents via pop-ups. *arXiv preprint arXiv:2411.02391*, 2024c.

Yuyang Zhang, Kangjie Chen, Xudong Jiang, Yuxiang Sun, Run Wang, and Lina Wang. Towards action hijacking of large language model-based agent. *arXiv preprint arXiv:2412.10807*, 2024d.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Shufang Zhu, Lucas M Tabajara, Jianwen Li, Geguang Pu, and Moshe Y Vardi. Symbolic ltlf synthesis. *arXiv preprint arXiv:1705.08426*, 2017.

A DETAILED INTRODUCTION TO SHIELDAGENT

A.1 NOTATIONS

Let $\mathcal{X}$ denote the environment, and let π_{agent} be the action policy of an agent we aim to shield. At each step i, the agent receives an observation $o_i \in \mathcal{X}$ and maps it to a partial state $s_i = f(o_i)$ via a state-space mapping function f. Specifically for web agents, f extracts accessibility trees (*AX-trees*) from the webpage's HTML and visual screenshots, condensing key information from lengthy observations Zhou et al. (2023). Then, the agent generates an action a_i by sampling from policy $a_i \sim \pi_{\text{agent}}(s_i)$ and progressively interacts with the environment $\mathcal{X}$.

A.2 SOLUTION SPACE

Given the uniqueness of verifying agent trajectories, we further categorize the predicates into two types: (1) **action predicate** p_a: indicates the action to be executed (e.g. *delete_data*); and (2) **state predicate** p_s: describes the environment states involved for specifying the condition that certain actions should be executed (e.g. *is_private*). A detailed explanation can be found in Appendix C.3.

Consequently, we characterize the solution space of LLM-based agents with the following two types of rules.

Action rule: an action rule ϕ_a specifies whether an action p_a should be executed or not under certain permissive or preventive conditions p_c. Note ϕ_a must involve at least one p_a. For example, the deletion action cannot be executed without user consent (i.e., $\neg is_user_authorized \rightarrow \neg delete_data$).

Physical rule: a physical rule ϕ_p specifies the natural constraints of the system, where conditions can logically depend on the others. For example, if a dataset contains private information then it should be classified as *red data* under GitLab's policy (i.e., $is_private \rightarrow is_red_data$).

Since predicates can sometimes be inaccurately assigned, ϕ_p can serve as knowledge in ASPM to enhance the robustness of our shield Kang & Li (2024). With these rules, SHIELDAGENT can effectively reason in the solution space to shield the agent action with high accuracy and robustness.

B ADDITIONAL RESULTS

B.1 ST-WEBAGENTBENCH

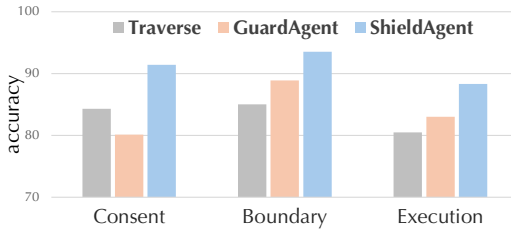

Figure 3: Performance comparison of SHIELDAGENT with *rule traverse* and *GuardAgent* baselines on ST-WebAgentBench. We report the individual guardrail accuracy for each risk category.

Table 4: Comparison of guardrail performance across three risk categories in ST-WebAgentBench Levy et al. (2024). Specifically, we report the averaged accuracy (ACC) and false positive rate (FPR) for each evaluation category, along with overall averages. The best performance is in bold.

Guardrail	User Consent		Boundary		Strict Execution		Overall	
	ACC ↑	FPR ↓	ACC ↑	FPR ↓	ACC ↑	FPR ↓	ACC ↑	FPR ↓
Direct	78.0	5.0	72.3	**3.4**	71.9	**4.3**	74.1	4.2
Rule Traverse	84.3	10.7	85.0	11.5	80.5	7.0	83.3	9.7
GuardAgent	80.1	4.5	88.9	8.7	83.0	6.5	84.0	6.6
SHIELDAGENT	**91.4**	**4.2**	**93.5**	4.0	**88.3**	5.1	**91.1**	**4.4**

B.2 VWA-ADV

Specifically, VWA-Adv Wu et al. (2025) attacks web agents by perturbing either the text instruction by adding a suffix or the image input by adding a bounded noise. Specifically, VWA-Adv constructs 200 diverse risky instructions based on the three environments from VisualWebArena Koh et al. (2024). The environments are detailed as follows:

Classifieds. Classifieds is a similar environment inspired by real-world platforms like Craigslist and Facebook Marketplace, comprising roughly 66K listings and uses OSClass—an open-source content management system—allowing realistic tasks such as posting, searching, commenting, and reviewing.

Shopping. This environment builds on the e-commerce site from WebArena Zhou et al. (2023), where successful navigation requires both textual and visual comprehension of product images, reflecting typical online shopping tasks.

Reddit. Adopting the social forum environment from WebArena, this environment hosts 31K+ posts (including images and memes) across different subreddits. The content variety offers broad coverage of social media interactions and challenges in forum-based tasks.

Table 5: Guardrail performance comparison on **VWA-Adv** Wu et al. (2025) across three environments in VisualWebArena Koh et al. (2024), i.e., *Classifieds*, *Reddit*, *Shopping*, under two perturbation sources, i.e., *text-based* and *image-based*. We report accuracy (ACC) and false positive rate (FPR) for each environment. The best performance is in bold.

Perturbation Source	Guardrail	Classifieds		Reddit		Shopping		Overall	
		ACC ↑	FPR ↓	ACC ↑	FPR ↓	ACC ↑	FPR ↓	ACC ↑	FPR ↓
Text-based	Direct	87.8	4.6	91.1	3.9	90.1	5.0	89.7	4.5
	GuardAgent	90.5	6.8	87.3	**2.6**	91.8	5.8	89.9	5.1
	SHIELDAGENT	**93.2**	**3.4**	**93.4**	4.9	**95.1**	**3.2**	**93.9**	**3.8**
Image-based	Direct	**93.7**	3.5	91.2	4.3	87.9	3.6	90.9	3.8
	GuardAgent	92.4	3.9	87.2	**2.7**	90.0	4.1	89.9	3.6
	SHIELDAGENT	91.0	**3.4**	**96.6**	**2.7**	**94.9**	**3.0**	**94.2**	**3.0**

B.3 AGENTHARM

Table 6: Guardrail performance comparison on **AgentHarm** Andriushchenko et al. across 11 harm categories. The reported numbers have been slightly perturbed (by ± 0.1 in an alternating pattern) and the overall is recalculated as the arithmetic mean of the 11 categories. The best performance is in bold.

		Fraud	Cybercrime	Self-harm	Harassment	Sexual	Copyright	Drugs	Disinfo.	Hate	Violence	Terrorism	Overall
Direct	**ACC**	75.7	82.4	76.5	80.6	82.2	72.0	**82.0**	76.9	71.0	75.8	71.1	76.9
	FPR	5.2	**3.6**	**3.6**	3.8	**3.8**	3.9	7.0	4.1	**3.5**	4.4	5.1	4.4
GuardAgent	**ACC**	82.6	66.1	75.1	75.9	82.1	69.6	76.6	80.1	77.7	**92.4**	83.9	78.4
	FPR	4.7	4.0	4.5	3.4	6.3	4.3	**3.8**	**3.2**	3.7	**3.3**	4.2	4.1
SHIELDAGENT	**ACC**	**89.1**	**92.9**	**82.5**	**92.4**	**94.0**	**89.0**	80.4	**81.9**	**81.7**	83.9	**88.3**	**86.9**
	FPR	**4.6**	4.9	3.9	**2.5**	4.0	**2.1**	5.5	4.2	3.8	4.7	**3.2**	**3.9**

C ACTION-BASED PROBABILISTIC SAFETY POLICY MODEL

C.1 AUTOMATED POLICY EXTRACTION

We detail the prompt for automated policy extraction in Appendix H and LTL rule extraction in Appendix H.

C.2 SAFETY POLICY MODEL CONSTRUCTION

C.2.1 AUTOMATIC POLICY AND RULE EXTRACTION

Specifically, we detail the prompt used for extracting structured policies in Appendix H). Specifically, each policy contains the following four elements:

1. **Term definition**: clearly defines all the terms used for specifying the policy, such that each policy block can be interpreted independently without any ambiguity.
2. **Application scope**: specifies the conditions (e.g. time period, user group, region) under which the policy applies.
3. **Policy description**: specifies the exact regulatory constraint or guideline (e.g. *allowable* and *non-allowable* actions).
4. **Reference**: lists original document source where the policy is extracted from, such that maintainers can easily trace them back for verifiability.

C.3 LINEAR TEMPORAL LOGIC (LTL) RULES

Temporal logic represents propositional and first-order logical reasoning with respect to time. *Linear temporal logic over finite traces* (LTL_f) Zhu et al. (2017) is a form of temporal logic that deals with finite sequences, i.e., finite-length trajectories.

Syntax. The syntax of an LTL_f formula φ over a set of propositional variables P is defined as:

$$\varphi ::= p \in P \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \Box\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2. \tag{7}$$

Specifically, LTL_f formulas include all standard propositional connectives: *AND* ($\wedge$), *OR* ($\vee$), *XOR* ($\oplus$), *NOT* ($\neg$), *IMPLY* ($\rightarrow$), and so on. They also use the following temporal operators (interpreted over finite traces):

- **Always** ($\Box\varphi_1$): φ_1 is true at every step in the finite trajectory.
- **Sometimes** ($\Diamond\varphi_1$): φ_1 is true at least once in the finite trajectory.
- **Next** ($\bigcirc\,\varphi_1$): φ_1 is true in the next step.
- **Until** ($\varphi_1 \,\mathcal{U}\, \varphi_2$): φ_1 must hold true at each step until (and including) the step when φ_2 first becomes true. In a finite trace, φ_2 must become true at some future step.

Specifically, φ_1 and φ_2 are themselves LTL_f formulas. An LTL_f formula is composed of variables in P and logic operations specified above.

Trajectory. A finite sequence of truth assignments to variables in P is called a *trajectory*. Let Φ denote a set of LTL_f specifications (i.e., $\{\phi \mid \phi \in \Phi\}$), we have $\zeta \models \Phi$ to denote that a trajectory ζ satisfies the LTL_f specification Φ.

C.4 ASPM STRUCTURE OPTIMIZATION

We detail the prompt for the verifiability refinement of ASPM in Appendix H and redundancy merging in Appendix H.

We detail the overall procedure of the iterative ASPM structure optimization in Algorithm 2.

C.5 TRAINING ASPM

Algorithm 2 ASPM Structure Optimization

Require: Predicate set $\mathcal{P} = \{\mathcal{P}_a, \mathcal{P}_s\}$; Rule set $\mathcal{R} = \{\mathcal{R}_a, \mathcal{R}_p\}$; Embedding model $\mathcal{E}$; Clustering algorithm $\mathcal{C}$; Refinement budget N_{b}; Max iterations M_{it}; Surrogate LLM; Graph $G = (\mathcal{P}, E)$ with initial edge weights E.

Initialize vagueness score for each predicate $\mathcal{V}_p, p \in \mathcal{P}$ ▷ Calculate via Eq. (3)
$\mathcal{V}_r = \max\{\mathcal{V}_{p_1}, \ldots, \mathcal{V}_{p_{|\mathcal{P}_r|}}\}, \mathcal{P}_r \subseteq \mathcal{P}$ ▷ Compute vagueness score for each rule
Initialize a max-heap $\mathcal{U} \leftarrow \{(\mathcal{V}_r, r) \mid r \in \mathcal{R}\}$
$n \leftarrow 0$ ▷ Count how many refinements have been done
for $m = 1$ to M_{it} **do**
  changed ← false ▷ Tracks if any update occurred in this iteration
  while $\mathcal{U} \neq \emptyset \wedge n \leq N_{\text{b}}$ **do**
    $(_, r) \leftarrow \text{HeapPop}(\mathcal{U})$ ▷ Pop the most *vague* rule
    if LLM_verifiable(r) = false **then**
      $r_{\text{new}} \leftarrow \text{LLM_refine}(r, \mathcal{P}_r)$ ▷ Refine rule r to be *verifiable*; update its predicates if needed
      Update $\mathcal{R}$: replace r with r_{new}
      Update $\mathcal{P}$: if r_{new} introduces or revises predicates
      Recompute $\mathcal{V}_p$ for any changed predicate p in r_{new}
      Recompute $\mathcal{V}_{r_{\text{new}}} = \max\{\mathcal{V}_p \mid p \in \mathcal{P}_{r_{\text{new}}}\}$
      Push $(\mathcal{V}_{r_{\text{new}}}, r_{\text{new}})$ into $\mathcal{U}$
      $n \leftarrow n + 1$
      changed ← true
    end if
  end while
  $\mathcal{K} \leftarrow \mathcal{C}(G)$ ▷ Cluster predicates in G to prune redundancy
  for each cluster $C \in \mathcal{K}$ **do**
    $p_{\text{merged}} \leftarrow \text{LLM_merge}(C, \mathcal{R})$ ▷ Merge similar predicates/rules in C if beneficial
    if $p_{\text{merged}} \neq \emptyset$ **then**
      Update G: add p_{merged}, remove predicates in C
      Update $\mathcal{R}$ to replace references of predicates in C with p_{merged}
      Recompute $\mathcal{V}_{p_{\text{merged}}}$ and any affected $\mathcal{V}_r$
      Push updated rules into $\mathcal{U}$ by their new $\mathcal{V}_r$
      changed ← true
    end if
  end for
  if changed = false **then**
    break ▷ No more refinements or merges
  end if
end for
return ASPM $\mathcal{G}_{\text{ASPM}}$ with optimized structure and randomized weights

D SHIELDAGENT FRAMEWORK

E SHIELDAGENT-BENCH

E.1 RISK CATEGORIES

We categorize the unsafe trajectories from SHIELDAGENT-BENCH into the following seven risk categories.

(1) **Access restriction**: Ensuring the agent only interacts with explicitly authorized areas within an application (e.g., enforcing user-specific access control); (2) **Content restriction**: Verifying that content handling follows predefined policies (e.g., preventing exposure of private or harmful data); (3) **Hallucination**: the cases where the agent generates or retrieves factually incorrect or misleading outputs in information-seeking tasks; (4) **Instruction adherence**: Assessing the agent's ability to strictly follow user-provided instructions and constraints without deviation; (5) **Operational restriction**: Enforcing explicit policy-based operational constraints, such as requiring user permission

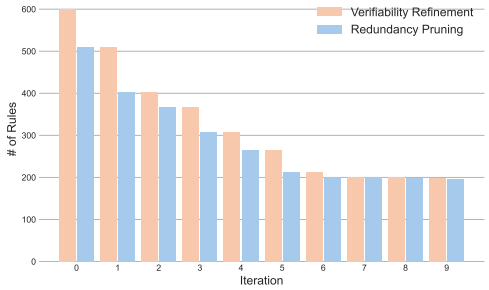

Figure 4: The number of rules during each iteration step for GitLab policy. Specifically, the orange bar denotes the number of rules after each *verifiability refinement* step, and the blue bar denotes the number of rules after each *redundancy pruning* step.

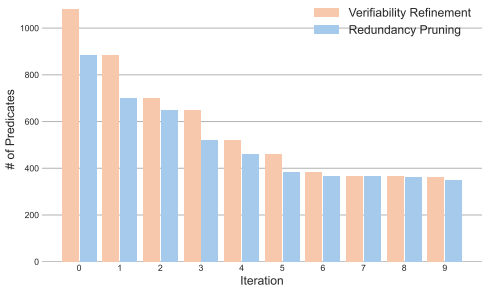

Figure 5: The number of predicates during each iteration step for GitLab policy. Specifically, the orange bar denotes the number of predicates after each *verifiability refinement* step, and the blue bar denotes the number of predicates after each *redundancy pruning* step.

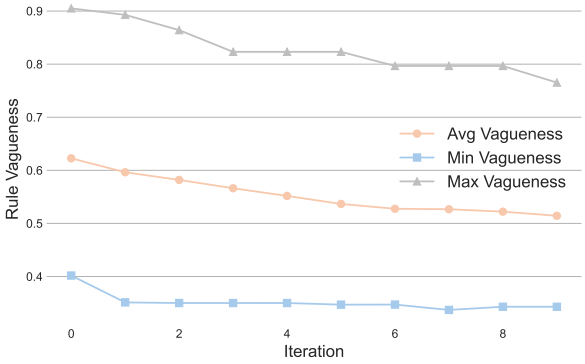

Figure 6: The vagueness score of the rule set during each iteration step for optimizing the GitLab policy. Specifically, we leverage GPT-4o as a judge and prompt it to evaluate the vagueness of each rule within the rule set. A lower vagueness score signifies that the rules are more concrete and therefore more easily verified.

before executing sensitive actions; (6) **Typical error pattern**: Identifying common failure patterns like infinite loops or redundant executions; (7) **Long-term risks**: Evaluating actions with delayed consequences, such as repeated failed login attempts leading to account lockout.

F Detailed Experiment Results

F.1 Dataset Distribution

We detail the distribution of samples in our proposed ShieldAgent-Bench dataset in Figure 7.

Table 7: Distribution of samples in our proposed ShieldAgent-Bench dataset. For each environment, we report the number of *safe* and *unsafe* trajectories. Each instruction is paired with one *safe* trajectory (i.e., compliant with all policies) and one *unsafe* trajectory (i.e., violating at least one policy), such that these paired trajectories are always equal in quantity.

Environment	Unsafe	Safe	Total
Shopping	265	265	530
CMS	260	260	520
Reddit	230	230	460
GitLab	450	450	900
Maps	160	160	320
SuiteCRM	190	190	380

Algorithm 3 ASPM TRAINING PIPELINE

Require: Rule set $\mathcal{R}$; *state predicates* $\mathcal{P}_s$ and *action predicates* $\mathcal{P}_a$; similarity threshold θ; number of clusters k.

$A \in \{0,1\}^{|\mathcal{P}_s| \times |\mathcal{P}_s|} \leftarrow \mathbf{0}$ ▷ Initialize adjacency matrix
$A_{ij} \leftarrow 1$ if (p_s^i, p_s^j) co-occur in any rule OR $\text{cosSim}\big(\text{emb}(p_s^i),\ \text{emb}(p_s^j)\big) \geq \theta$; else 0. ▷ Build adjacency matrix
labels $\leftarrow$ SPECTRALCLUSTERING$(A,\ k)$ ▷ Cluster the state predicates into k groups
for $\ell = 1$ **to** k **do**
  $C_p^\ell \leftarrow \{p_s \mid \text{labels}[p_s] = \ell\}$ ▷ Form predicate clusters $\mathcal{C}_p$
end for
for each pair (p_s^i, p_s^j) **that co-occur do**
  if $\text{labels}[p_s^i] \neq \text{labels}[p_s^j]$ **then**
    $\mathcal{C}_p^\ell \leftarrow \mathcal{C}_p^\ell \cup \mathcal{C}_p^m$ s.t. $p_s^i \in \mathcal{C}_p^\ell, p_s^j \in \mathcal{C}_p^m$ ▷ If two co-occurring predicates appear in different clusters, merge them
  end if
end for
for $\ell = 1$ **to** k' **do**
  $C_r^\ell \leftarrow \{r_s \mid p_s \in C_p^\ell\}$ ▷ Group rules which share state predicates in the same cluster
end for
$\mathcal{G}_{\text{ASPM}} \leftarrow \varnothing$ ▷ Initialize ASPM as an empty dictionary with actions as keys
for each $p_a \in \mathcal{P}_a$ **do**
  for each rule cluster $C_r^\ell \in \mathcal{C}_r$ **do**
    for each rule $r \in C_r^\ell$ **do**
      if $p_a^r \in r$ **then**
        $\mathcal{G}_{\text{ASPM}}[p_a] = \mathcal{G}_{\text{ASPM}}[p_a] \cup C_r^\ell$ ▷ Associate action circuits with any relevant rule clusters
        break
      end if
    end for
  end for
end for
for each action circuit $\mathcal{C}_{\theta_a}^{p_a}$ **do**
  for each rule $r \in \mathcal{C}_{\theta_a}^{p_a}$ **do**
    Initialize rule weight θ_r randomly
  end for
  for epoch $= 1$ **to** max epochs **do**
    for $i = 1$ **to** N **do**
      Compute $P_\theta(\mu_{p_a=1}^{(i)})$ and $P_\theta(\mu_{p_a=0}^{(i)})$ ▷ Run probabilistic inference to obtain corresponding safety probabilities via Eq. (4)
      Compute loss $\mathcal{L}(\theta)$ ▷ Calculate loss w.r.t. the groundtruth labels via Eq. (6)
      Update θ using gradient descent
    end for
  end for
end for
return Action-based safety policy model $\mathcal{G}_{\text{ASPM}}$ with trained weights

F.2 COMPARISON WITH EXISTING DATASETS

G CASE STUDY

Table 8: Comparison of SHIELDAGENT-BENCH with existing agent safety benchmarks. SHIELDAGENT-BENCH extends prior work by offering more samples, operation risk categories, and types of adversarial perturbations (both *agent-based* and *environment-based*). In addition, SHIELDAGENT-BENCH provides verified annotations of both risky inputs and output trajectories, explicitly defining each case of safety violations, and annotating relevant policies for verifying each trajectory.

Benchmark	#Sample	#Operation Risk	#Attack Type	#Environment	Risky Trajectory	Risk Explanation	#Rule
ST-Web Levy et al. (2024)	234	3	0	3		✓	36
AgentHarm Andriushchenko et al.	440	1	0	0			0
VWA-Adv Wu et al. (2025)	200	1	1	3			0
SHIELDAGENT-BENCH	3110	7	2	6	✓	✓	1080

GitLab – **Access Restriction**

Policy

- **Policy description**: Users must not access, modify, delete, or disseminate GitLab's or its users' confidential information without authorization.
- **Applicable scope**: Applicable to all GitLab and user data handled via GitLab services.
- **Term definition**: Confidential Information: Unauthorized access, modification, or dissemination of protected data.
- **Policy source**: handbook.LACA.acceptable_use_policy

Rule

- **Predicates**:
  - *is_confidential_data*:
    - description: The data is classified as confidential information
    - type: **condition**
  - *user_authorized*:
    - description: The user has authorized to perform actions on the data.
    - type: **condition**
  - *delete_data*:
    - description: The user deletes the data.
    - type: **action**
- **Rule Logic**: (is_confidential_data AND NOT is_authorized_user) IMPLIES NOT delete_data.
- **Logic template**: (P1 AND NOT P2) IMPLIES NOT P3
- **Risk type**: access restriction; content restriction

Figure 7: An example of the *access restriction* policy and the corresponding extracted rules from GitLab policy handbook.

GitLab – **Content Restriction**

Policy

- **Policy description**: Users must not create, share, or facilitate the distribution of illegal content, including child exploitation materials.
- **Applicable scope**: Applicable to all content hosted or transmitted via GitLab.
- **Term definition**: (1) Illegal or Harmful Content: Content that promotes illegal activities or endangers safety; (2) Child Exploitation: Any content that depicts or facilitates the exploitation of minors.
- **Policy source**: handbook.content_security_policy

Rule

- **Predicates**:
  - *is_illegal_content*:
    - description: The content is classified as illegal or harmful.
    - type: **condition**
  - *create_content*:
    - description: An action to create content.
    - type: **action**
- **Rule Logic**: is_illegal_content IMPLIES NOT create_content
- **Logic template**: P1 IMPLIES NOT P2
- **Risk type**: content restriction

Figure 8: An example of the *content restriction* policy and the corresponding extracted rules from GitLab policy handbook.

GitLab – **Hallucination**

Policy

- **Policy description**: The retrieved information must be consistent with the information source and should not introduce fabricated content.
- **Applicable scope**: Applies to all retrieved content from GitLab services.
- **Term definition**: (1) Source Consistency: The retrieved content must align factually with the source data.
- **Policy source**: empirical_policy

Rule

- **Predicates**:
  - *consistent_with_source*:
    - description: The retrieved or produced content is consistent with the source data.
    - type: **condition**
  - *send_to_user*:
    - description: An action to send specific information/data to the user.
    - type: **action**
- **Rule Logic**: NOT consistent_with_source IMPLIES NOT *send_to_user*
- **Logic template**: NOT P1 IMPLIES NOT P2
- **Risk type**: hallucination

Figure 9: An example of the *hallucination* policy and the corresponding extracted rules from GitLab policy handbook.

H PROMPT TEMPLATE

Prompt Template for Video Guardrail

SYSTEM: You are a helpful policy extraction model to identify actionable policies from organizational safety guidelines. Your task is to exhaust all the potential policies from the provided organization handbook which sets restrictions or guidelines for user or entity behaviors in this organization. You will extract specific elements from the given guidelines to produce structured and actionable outputs.

USER: As a policy extraction model to clean up policies from {organization (e.g. GitLab)}, your tasks are:

1. Read and analyze the provided safety policies carefully, section by section.
2. Exhaust all actionable policies that are concrete and explicitly constrain behaviors.
3. For each policy, extract the following four elements:
    1. **Definition**: Any term definitions, boundaries, or interpretative descriptions for the policy to ensure it can be interpreted without any ambiguity. These definitions should be organized in a list.
    2. **Scope**: Conditions under which this policy is enforceable (e.g. time period, user group).
    3. **Policy Description**: The exact description of the policy detailing the restriction or guideline.
    4. **Reference**: All the referenced sources in the original policy article from which the policy elements were extracted. These sources should be organized piece by piece in a list.

Extraction Guidelines:

- Do not summarize, modify, or simplify any part of the original policy. Copy the exact descriptions.
- Ensure each extracted policy is self-contained and can be fully interpreted by looking at its **Definition**, **Scope**, and **Policy Description**.
- If the **Definition** or **Scope** is unclear, leave the value as **None**.
- Avoid grouping multiple policies into one block. Extract policies as individual pieces of statements.

Provide the output in the following JSON format:

```json
[
  {
    "definition": ["Exact term definition or interpretive description."],
    "scope": "Conditions under which the policy is enforceable.",
    "policy_description": "Exact description of the policy.",
    "reference": ["Original source where the elements were extracted."]
  },
    ...
]
```

Output Requirement:

- Each policy must focus on explicitly restricting or guiding behaviors.
- Ensure policies are actionable and clear.
- Do not combine unrelated statements into one policy block.

Question List for QA Guardrail Task

SYSTEM: You are an advanced policy translation model designed to convert organizational policies into structured Linear Temporal Logic (LTL) rules. Your task is to extract verifiable rules from the provided safety guidelines and express them in a machine-interpretable format while maintaining full compliance with logical correctness.

USER: As a policy-to-LTL conversion model, your tasks are:

1. Carefully analyze the policy's **definition**, **scope**, and **policy description**.
2. Break down the policy into structured rules that precisely capture its constraints and requirements.
3. Translate each rule into LTL using atomic predicates derived from the policy.

Translation Guidelines:

- Use **atomic predicates** that are directly verifiable from the agent's observations and action history.
- Prefer **positive predicates** over negative ones (e.g., use `store_data` instead of `is_data_stored`).
- If a rule involves multiple predicates, decompose it into smaller, verifiable atomic rules whenever possible.
- Emphasize **action-based predicates**, ensuring that constrained actions are positioned appropriately within logical expressions (e.g., "only authorized users can access personal data" should be expressed as:

$$(\texttt{is_authorized} \wedge \texttt{has_legitimate_need}) \Rightarrow \texttt{access_personal_data} \tag{8}$$

).

Predicate Formatting: Each predicate must include:

- **Predicate Name**: Use snake_case format.
- **Description**: A brief, clear explanation of what the predicate represents.
- **Keywords**: A list of descriptive keywords providing relevant context (e.g., actions, entities, attributes).

LTL Symbol Definitions:

- **Always**: `ALWAYS`
- **Eventually**: `EVENTUALLY`
- **Next**: `NEXT`
- **Until**: `UNTIL`
- **Not**: `NOT`
- **And**: `AND`
- **Or**: `OR`
- **Implies**: `IMPLIES`

Output Format: ```json

```
[
  {
    "predicates": [
      ["predicate_name", "Description of the predicate.", ["kw1", "kw2", ...]]
    ],
    "logic": "LTL rule using predicate names."
  },
  ...
]
```

Output Requirements:

- Ensure each rule is explicitly defined and unambiguous.
- Keep predicates general when applicable (e.g., use `create_project` instead of `click_create_project`).
- Avoid combining unrelated rules into a single LTL statement.

A List of Examples for Policy Guidelines

SYSTEM: You are a helpful predicate refinement model tasked with ensuring predicates in the corresponding rules are clean, verifiable, concrete, and accurate enough to represent the safety policies. Your task is to verify each predicate and refine or remove it if necessary.

USER: As a predicate refinement model, your tasks are:

1. Check if the provided predicate satisfies the following criteria:
   - **Verifiable**: It should be directly verifiable from the agent's observation or action history.
   - **Concrete**: It should be specific and unambiguous.
   - **Accurate**: It must represent the intended fact or condition precisely.
   - **Atomic**: It should describe only one fact or action. If it combines multiple facts, break it into smaller predicates.
   - **Necessary**: The predicate must refer to meaningful information. If it is redundant or assumed by default, remove it.
   - **Unambiguous**: If the same predicate name is used in different rules but has different meanings, rename it for clarity.
2. If refinement is needed, refine the predicate accordingly with one of the following:
   - Rewrite the predicate if it is unclear or inaccurate.
   - Break it down into smaller atomic predicates if it combines multiple facts or conditions.
   - Rename the predicate to reflect its context if it is ambiguous.
   - Remove the predicate if it is redundant or unnecessary for the rule.

Output Requirements:

- Provide step-by-step reasoning under the section **Reasoning**.
- Include the label on whether the predicate is **good**, **needs refinement**, or **redundant**.
- If refinement is needed, provide a structured JSON including:
  - Updated predicate with definitions and keywords.
  - Each of the updated rules which are associated with the updated predicate.
  - Definitions of the predicate in each rule's context.

Output Format:
Reasoning:
1. Step-by-step reasoning for why the predicate is good, needs refinement, or is redundant.
2. If yes, then reason about how to refine or remove the redundant predicate.
Decision: Yes/No
If yes, then provide the following:
Output JSON:

```
{
  "rules": [
    {
      "predicates": [
        ["predicate_name", "Predicate definition.", ["keywords"]]
      ],
      "logic": "logic_expression_involving_predicates"
    }
  ]
}
```

{Few-shot Examples}

Policy Guidelines for Unseen Categories

SYSTEM: You are a helpful predicate merging model tasked with analyzing a collection of similar predicates and their associated rules to identify whether there are at least predicates that can be merged or pruned. Your goal is to simplify and unify rule representation while ensuring the meaning and completeness of the rules remain intact after modifying the predicates.

USER: As a predicate merging model, your tasks are:

1. Identify predicates in the cluster that can be merged based on the following conditions:
   - **Redundant Predicates**: If two or more predicates describe the same action or condition but use different names or phrasing, merge them into one.
   - **Identical Rule Semantics**: If two rules describe the same behavior or restriction but are phrased differently, unify the predicates and merge their logics to represent them with fewer rules.
2. Ensure the merged predicates satisfy the following:
   - **Consistency**: The merged predicate must be meaningful and represent the combined intent of the original predicates.
   - **Completeness**: The new rules must perfectly preserve the logic and intent of all original rules.

Output Requirements:

- Provide step-by-step reasoning under the section **Reasoning**.
- Include a decision label on whether the predicates should be merged.
- If merging is needed, provide a structured JSON including:
  - Updated predicates with definitions and keywords.
  - Updated rules with the new merged predicates.

Output Format:
Reasoning:

1. Step-by-step reasoning for why the predicates should or should not be merged.
2. If merging is needed, explain how the predicates and rules were updated to ensure completeness and consistency.

Decision: Yes/No
If yes, then provide the following:
Output JSON:

```
{
  "rules": [
    {
      "predicates": [
        ["predicate_name", "Predicate definition.", ["keywords"]]
      ],
      "logic": "logic_expression_involving_predicates"
    }
  ]
}
```

Few-shot Examples