CONTEXTGUARD: CONTEXT-CONDITIONED NEURO-SYMBOLIC SAFETY GUARDRAIL FOR FINE-GRAINED INTERVENTION IN MOBILE GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mobile GUI agents increasingly perform complex tasks autonomously, making runtime safety mechanisms that account for the evolving interaction context increasingly important. Existing guardrails typically either use context as auxiliary information for model-based risk prediction or enforce predefined safety rules over fixed representations. While model-based approaches can capture contextual risks, their safety judgments may rely on opaque external models and incur additional computational cost, whereas rule-based approaches provide explicit constraints but lack a unified mechanism for grounding evolving interaction context into constraint applicability. We present **ContextGuard**, a new context-conditioned neuro-symbolic runtime safety guardrail that makes interaction context part of the safety semantics itself. ContextGuard grounds heterogeneous mobile observations into soft semantic predicates capturing interface and action semantics as well as task-dependent and history-dependent context, and incorporates these predicates directly into explicit safety constraints. A neuro-symbolic reasoning module evaluates explicit safety constraints over these predicates, allowing contextual information to directly determine which constraints apply while retaining inspectable constraint-level reasoning. The resulting decisions are mapped to three runtime interventions: `PERMIT`, `CONFIRM`, and `STOP`. On a diverse evaluation set of 592 task trajectories spanning 10 real mobile applications and 10 safety risk categories under the AndroidLab evaluation setting, ContextGuard achieves 90.9% Binary Accuracy, 4.0% False Positive Rate, and 87.8% Intervention Accuracy, achieving the highest fine-grained Intervention Accuracy among the evaluated baselines. Ablation results show that incorporating task instructions and execution history improves Intervention Accuracy from 56.6% to 87.8%, while adding execution history introduces only 0.11 ms/task of additional guardrail overhead. These results demonstrate that ContextGuard integrates contextual semantics directly into safety constraints, combines flexible semantic grounding with interpretable neuro-symbolic reasoning, and maintains practical guardrail overhead for mobile-agent safety intervention. The source code is available at `https://anonymous.4open.science/r/ContextGuard/`.

1 INTRODUCTION

Large language models (LLMs) (Achiam et al., 2023; Bai et al., 2023; Comanici et al., 2025; Guo et al., 2025) have enabled autonomous agents to perform increasingly complex tasks through graphical user interfaces (GUIs), including information retrieval, application interaction, and multi-step task execution (Wang et al., 2024; Zhang et al., 2025; Qin et al., 2025; Yehudai et al., 2026). As GUI agents become capable of executing longer sequences of actions with limited human intervention, ensuring that their interactions remain safe and consistent with user intent becomes increasingly important (Zhang et al., 2024; Kuntz et al., 2026). An erroneous action, such as submitting unintended information, performing an irreversible operation, or repeatedly executing a high-impact action after a task has been completed, can lead to undesirable consequences. Runtime safety guardrails (Wang et al., 2026; Liu et al., 2026; Ravichandran et al., 2026; Yu et al., 2025; Huang et al., 2026b; Xiang et al., 2025; Luo et al., 2025; Hu et al., 2025) therefore provide an important layer of protection by monitoring and intervening on agent actions before they are executed.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Existing safety guardrails for GUI agents predominantly formulate safety monitoring as action- or trajectory-level risk detection: given an interface state, task instruction, and/or proposed action, the guardrail predicts whether the current behavior is safe or risky. These approaches can be broadly viewed as model-based or rule-based guardrails. Model-based approaches increasingly incorporate task and interaction context into safety judgments; for example, OS-Sentinel (Sun et al., 2026) combines formal verification with a VLM-based contextual judge for mobile-agent safety, while AgentAuditor (Luo et al., 2026) uses memory-augmented context-aware retrieval to improve LLM-based safety evaluation. Such approaches demonstrate the value of contextual information, but context is primarily provided as additional evidence to a model-based risk judgment, rather than being represented as part of the conditions that determine when a safety constraint applies. Moreover, model-based judgments generally provide limited insight into how contextual evidence leads to a particular safety decision. Rule-based approaches (Wang et al., 2026; Shi et al., 2025; Chen et al., 2025) instead encode safety requirements as explicit predicates and executable policies. While these approaches provide transparent and enforceable constraints, they typically operate over predefined symbolic representations and do not provide a unified mechanism for grounding uncertain mobile-interface semantics together with evolving task context. Finally, context-aware guardrails that rely on external LLM or VLM inference introduce additional runtime computation for contextual safety evaluation, which can be undesirable for step-level intervention during interactive execution. These limitations motivate a different design in which **context becomes part of the safety semantics itself**, rather than merely serving as auxiliary evidence for risk prediction. Such a design should connect contextual understanding with explicit safety constraints, while retaining the transparency of symbolic reasoning and avoiding the repeated use of expensive external models during runtime verification. This motivates a neuro-symbolic approach (Badreddine et al., 2022; Manhaeve et al., 2021; De Smet et al., 2023; Pryor et al., 2023) that combines flexible semantic interpretation with explicit and auditable safety reasoning, enabling contextual safety decisions to remain both interpretable and practical for runtime deployment.

To address these limitations, we introduce **ContextGuard**, a new context-conditioned neuro-symbolic runtime guardrail for mobile agents. As illustrated in Figure 1, ContextGuard bridges contextual semantic understanding and explicit safety enforcement through a unified neuro-symbolic framework. It first grounds heterogeneous mobile interaction observations into a shared set of soft semantic predicates capturing both the current interface and the evolving interaction context. These predicates describe properties such as task alignment, task overreach, commitment boundaries, external effects, irreversibility, task completion, and repeated high-impact actions. The resulting semantic representation is then combined with a safety knowledge base containing explicit safety constraints and their contextual applicability conditions. A differentiable neuro-symbolic reasoner evaluates these constraints against the grounded predicates, allowing the applicable safety requirements to be determined according to both the current interface semantics and the evolving interaction context. ContextGuard then enforces the resulting safety decisions at runtime, either permitting the action, requesting confirmation, or stopping execution. Importantly, the neuro-symbolic formulation does not collapse contextual safety reasoning into an opaque risk score: neural grounding provides uncertain semantic evidence, while explicit safety constraints specify the conditions and intervention consequences that can be inspected and traced at runtime. Consequently, the same action can receive different interventions when its task and interaction context change.

We evaluate ContextGuard on a diverse evaluation set of 592 task trajectories spanning 10 real mobile applications and 10 safety risk categories under the AndroidLab (Xu et al., 2025) evaluation setting. The evaluation considers both binary safety detection and fine-grained intervention decisions among `PERMIT`, `CONFIRM`, and `STOP`. ContextGuard achieves strong performance across both levels while maintaining practical runtime overhead, and controlled ablation examines the contribution of task and execution-history context.

In summary, our contributions are:

- We formulate mobile GUI-agent safety as a **context-conditioned runtime intervention problem**, where the applicability of a safety constraint depends jointly on interface semantics, task instructions, proposed actions, and interaction history. This formulation explicitly models how the safety meaning of an action changes with the evolving interaction context.
- We propose **ContextGuard**, a new neuro-symbolic runtime guardrail that grounds heterogeneous mobile observations into soft semantic predicates and connects them to explicit safety constraints

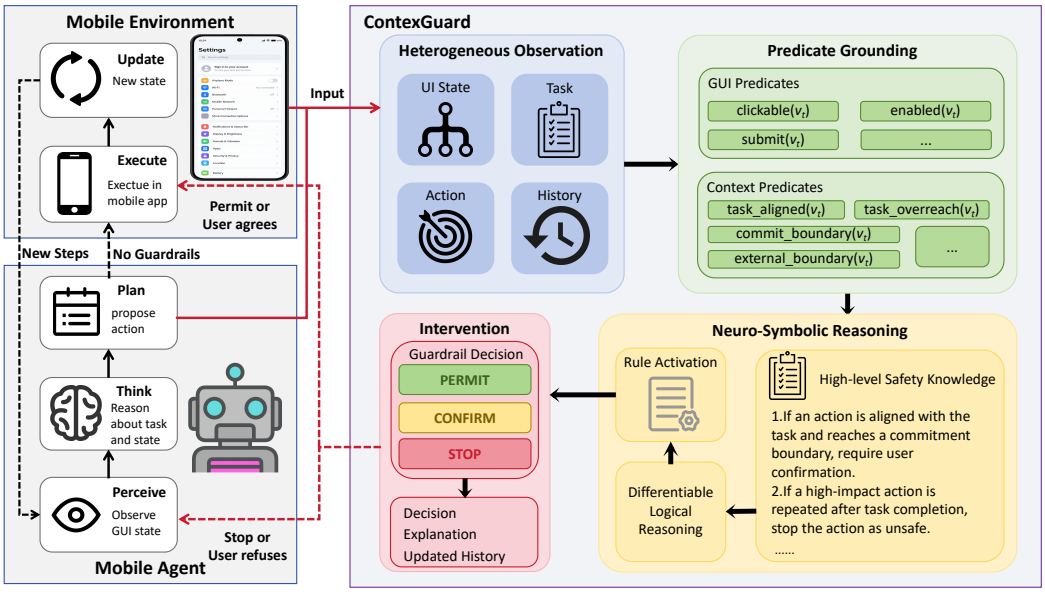


Figure 1: Overview of ContextGuard. The heterogeneous mobile interaction observation is grounded into soft interface/action and contextual predicates. These predicates are jointly evaluated against explicit context-conditioned safety constraints, whose applicability depends on the current task and interaction state. The resulting constraint activations are mapped to runtime interventions that permit, require confirmation, or stop the proposed action.

through neuro-symbolic reasoning. This design combines neural semantic flexibility with explicit, inspectable safety knowledge, allowing contextual semantics to directly determine which safety constraints apply and why an intervention is required.

- We conduct a comprehensive evaluation on 592 task trajectories spanning 10 real mobile applications and 10 safety risk categories under the AndroidLab (Xu et al., 2025) evaluation setting, covering both binary safety detection and **fine-grained intervention decisions** among `PERMIT`, `CONFIRM` and `STOP`. Experiments with controlled context ablation demonstrate the complementary value of task instructions and execution history for context-dependent intervention, while runtime measurements show that the complete context-conditioned guardrail incurs only a small additional computational cost.

2 Related Work

2.1 Neuro-Symbolic Reasoning for Agent Safety

Neuro-symbolic methods combine flexible neural representations with explicit symbolic knowledge, and have been explored in structured reasoning, embodied intelligence, planning, and safety verification (Manhaeve et al., 2018; De Smet et al., 2023; Badreddine et al., 2022; Manhaeve et al., 2021; Derkinderen et al., 2024; Pryor et al., 2023). Recent work has also applied neuro-symbolic reasoning to runtime safety guarding, such as using structured driving scenes and explicit traffic rules to constrain autonomous-driving actions (Idarraga et al., 2026). These studies demonstrate the value of combining uncertain semantic representations with explicit and interpretable safety constraints. However, existing neuro-symbolic safety approaches primarily focus on physical environments or agent decision making, where safety constraints are defined over relatively stable world states or domain-specific symbolic representations. Mobile GUI agents introduce a different form of safety reasoning: the meaning of an interaction can evolve with the user's task and the preceding workflow, even when the interface-level action remains similar. ContextGuard extends neuro-symbolic safety reasoning to this setting by incorporating task and interaction context into the semantic representation used for safety verification.

2.2 Safety Guardrails for GUI Agents

Safety guardrails for GUI agents can broadly be divided into rule-based and model-based approaches. Rule-based systems encode explicit safety specifications for runtime enforcement (Wang et al., 2026; Shi et al., 2025; Chen et al., 2025), while model-based approaches such as Safiron (Huang et al., 2026a) perform pre-execution risk detection and categorization. For mobile GUI agents, SeerGuard (Yu et al., 2026) uses a safety-augmented world model to predict action consequences, OS-Sentinel (Sun et al., 2026) combines formal verification with a VLM-based contextual judge, and AgentAuditor (Luo et al., 2026) incorporates contextual information through memory-augmented retrieval for LLM-based safety evaluation. Related safety benchmarks (Yin et al., 2024; Zhang et al., 2024; Kuntz et al., 2026; Chen et al., 2026) further emphasize the need to evaluate safety across diverse agent behaviors and interaction environments. Existing guardrails primarily use context for risk prediction, evidence retrieval, or rule-based enforcement, rather than as part of the conditions governing intervention. ContextGuard instead formulates safety guarding as a context-conditioned neuro-symbolic problem, where task- and history-dependent predicates determine the applicability of explicit safety constraints and thereby support context-dependent `PERMIT`, `CONFIRM`, and `STOP` decisions.

3 Methodology

3.1 Problem Formulation and System Overview

We consider an autonomous mobile GUI agent that interacts with a mobile application through a sequence of GUI actions. At time t, the agent receives a task instruction $\mathcal{T}$, observes the current mobile interface U_t, and proposes an action a_t. The preceding interaction trajectory is denoted by $\mathcal{H}_t$. ContextGuard operates alongside the agent as a runtime safety guardrail: before a proposed action is executed, it evaluates the action together with the current interaction context and produces an intervention decision $d_t \in \texttt{PERMIT}, \texttt{CONFIRM}, \texttt{STOP}$.

The architecture of ContextGuard consists of three logically distinct components, as illustrated in Figure 1: neural semantic grounding, neuro-symbolic safety reasoning, and runtime intervention. First, the neural grounding module transforms heterogeneous mobile interaction observations into a shared set of soft semantic predicates describing the current interface, proposed action, task instruction, and interaction history. Second, the neuro-symbolic reasoning module applies the explicit safety constraints encoded in a safety knowledge base to the grounded predicates, combining neural predicate confidence with symbolic constraint reasoning. The constraints specify symbolic conditions that may jointly involve interface semantics and contextual information, and the reasoning module aggregates the confidence of grounded predicates according to these conditions to determine which constraints are activated for the current action. Finally, the runtime intervention layer maps the applicable constraints to an executable decision and records the corresponding reasoning trace. This architecture separates semantic grounding, safety knowledge, and runtime enforcement, while using neuro-symbolic reasoning to connect grounded semantics with explicit safety constraints.

Let $O_t = (U_t, \mathcal{T}, a_t, \mathcal{H}_t)$ denote the heterogeneous observation available at step t. The neural grounding function ϕ_θ maps O_t to a set of soft semantic predicates $\mathcal{P}_t = \phi_\theta(O_t)$, which consists of interface and action predicates $\mathcal{P}_t^{ia}$ and contextual predicates $\mathcal{P}_t^{ctx}$. ContextGuard then applies an explicit safety knowledge base $\mathcal{K}$ to the grounded predicates through a neuro-symbolic reasoner $\mathcal{R}$, which determines the safety constraints applicable to the current interaction and produces the activated constraint set $\mathcal{C}_t$. The runtime intervention layer maps $\mathcal{C}_t$ to an intervention decision $d_t \in \texttt{PERMIT}, \texttt{CONFIRM}, \texttt{STOP}$. The overall verification process can therefore be written as $O_t \xrightarrow{\phi_\theta} \mathcal{P}_t \xrightarrow{\mathcal{K},\mathcal{R}} \mathcal{C}_t \rightarrow d_t$. Here, ϕ_θ performs neural semantic grounding, $\mathcal{K}$ specifies the explicit safety constraints, $\mathcal{R}$ evaluates their contextual applicability using the grounded predicate confidences, and $\mathcal{C}_t$ contains the constraints activated for the current interaction state. After the intervention decision is made, the relevant execution outcome and verification information are incorporated into the interaction history, yielding $\mathcal{H}_{t+1}$ for subsequent verification.

Algorithm 1 Context-Conditioned Neuro-Symbolic Runtime Verification

Require: Mobile interface U_t, task instruction $\mathcal{T}$, proposed action a_t, history $\mathcal{H}_t$
Ensure: Intervention d_t and audit trace Π_t

```
Construct observation O_t ← (U_t, T, a_t, H_t)
Obtain soft semantic predicates P_t ← φ_θ(O_t)
C_t ← ∅
for each relevant safety constraint κ_k ∈ K do
    Compute activation confidence α_k ← R(κ_k, P_t)
    if α_k ≥ δ then
        Add κ_k to C_t
    end if
end for
Construct audit trace Π_t from P_t and C_t
if a prohibition constraint is active in C_t then
    d_t ← STOP
else if a confirmation constraint is active in C_t then
    d_t ← CONFIRM
else
    d_t ← PERMIT
end if
Update H_{t+1} with relevant interaction state and intervention outcome
return d_t, Π_t
```

3.2 Heterogeneous Mobile Interaction Representation

ContextGuard represents each guarded interaction through four complementary sources: the current mobile UI state U_t, task instruction $\mathcal{T}$, proposed action a_t, and interaction history $\mathcal{H}_t$. The resulting observation is denoted by $O_t = (U_t, \mathcal{T}, a_t, \mathcal{H}_t)$. These sources provide complementary information for contextual safety verification: the UI describes the current interaction state, the task specifies the intended objective, the proposed action determines the immediate operation, and the history captures how the interaction has evolved.

A neural grounding function transforms the heterogeneous observation into a shared semantic representation, written as $\mathcal{P}_t = \phi_\theta(O_t)$. Rather than directly predicting whether an action is safe or unsafe, the grounding module extracts safety-relevant semantic properties from the current interface, proposed action, task instruction, and interaction history. These properties are subsequently used by the context-conditioned safety constraints for symbolic safety reasoning. The resulting representation consists of soft semantic predicates $\mathcal{P}_t = \mathcal{P}_t^{ia} \cup \mathcal{P}_t^{ctx}$, where $\mathcal{P}_t^{ia}$ captures properties of the current interface and proposed action, while $\mathcal{P}_t^{ctx}$ captures safety-relevant properties of the ongoing interaction. The interface and action predicates describe properties such as whether the operation is consequential, destructive, privacy-sensitive, or externally effective, and may additionally characterize the semantic properties of the interface entity affected by the action. The contextual predicates describe properties such as task alignment, task overreach, commitment boundaries, task completion, and repeated operations. For example, a tap targeting a "Submit" control can be grounded into semantic predicates indicating a submission operation or a financially consequential operation. Each predicate p_i is associated with a confidence value $\mu_i \in [0, 1]$, allowing uncertainty in semantic interpretation to be propagated to the subsequent neuro-symbolic reasoning stage. Further details on the predicate semantics and grounding of interface/action and contextual predicates are provided in Appendix A.1.

3.3 Context-Conditioned Neuro-Symbolic Safety Reasoning

Given the soft semantic predicates $\mathcal{P}_t$, the neuro-symbolic reasoner $\mathcal{R}$ evaluates the safety constraints encoded in the explicit knowledge base $\mathcal{K}$ to determine which constraints apply to the proposed action. Each safety constraint $\kappa_k \in \mathcal{K}$ specifies semantic conditions over the proposed operation and its interaction context. These conditions may jointly involve interface and action predicates as well as task- and history-dependent contextual predicates. Importantly, contextual predicates participate directly in the conditions of safety constraints, allowing the applicability of a

constraint to vary with the task and interaction state rather than treating context as auxiliary input to an independent risk classifier. Further details on the constraint formulation and representative context-conditioned safety constraints are provided in Appendix A.2.

Since the grounded predicates are associated with confidence values, the reasoner evaluates each relevant safety constraint using differentiable logical operators. For a constraint κ_k whose antecedent contains predicates $p_{k,1}, \ldots, p_{k,m}$ with confidence values $\mu_{k,1}, \ldots, \mu_{k,m}$, its activation confidence is computed as $\alpha_k = T(\mu_{k,1}, \ldots, \mu_{k,m})$, where $T(\cdot)$ denotes a differentiable t-norm implementing logical conjunction. The activated constraint set is then given by $\mathcal{C}_t = \{\kappa_k \in \mathcal{K} \mid \alpha_k \geq \delta\}$, where δ is the activation threshold. Thus, the reasoning process can be summarized as $\mathcal{P}_t \xrightarrow{\mathcal{K},\mathcal{R}} \mathcal{C}_t$: neural grounding provides uncertain semantic evidence, while the explicit safety constraints determine how this evidence is combined into runtime safety requirements. Further details on the differentiable constraint evaluation and activation procedure are provided in Appendix A.3.

This formulation combines the flexibility of neural semantic grounding with the transparency of symbolic safety reasoning. The neural component represents uncertain mobile-interface semantics as soft predicates, while the symbolic component explicitly specifies the conditions under which a safety constraint becomes applicable. Consequently, the reasoning trace can identify the semantic predicates and safety constraints supporting a non-permissive intervention, rather than reducing contextual safety assessment to an opaque risk score.

3.4 Runtime Intervention and Stateful Verification

Once the activated safety constraints $\mathcal{C}_t$ have been obtained, ContextGuard maps them to an executable runtime intervention. The guardrail produces one of three outcomes: `PERMIT`, `CONFIRM`, or `STOP`. `PERMIT` allows the proposed action to proceed when no applicable safety constraint requires intervention, `CONFIRM` pauses execution and requests explicit authorization when a confirmation constraint is active, and `STOP` blocks the action when a prohibition constraint is active. When multiple constraints are simultaneously activated, the most restrictive applicable intervention is selected according to the fixed precedence order `STOP` $>$ `CONFIRM` $>$ `PERMIT`. Thus, the intervention is determined directly by the activated safety constraints rather than by an additional learned decision model.

To support safety properties that depend on multiple interaction steps, ContextGuard maintains an evolving interaction history $\mathcal{H}_t$. After each guarded step, the history is updated with the relevant interaction state and intervention outcome, yielding $\mathcal{H}_{t+1}$ for subsequent verification. The updated history is provided to the neural grounding module in future steps, allowing contextual predicates to capture properties such as task progress, task completion, and repeated operations. This stateful design enables the applicability of safety constraints to evolve with the interaction without modifying the underlying agent policy. ContextGuard also records an audit trace Π_t linking grounded predicates, activated constraints, and the resulting intervention, providing decision provenance throughout runtime verification.

4 Experiment

4.1 Experimental Setup

Environment and Evaluation Protocol. We evaluate ContextGuard under an AndroidLab (Xu et al., 2025) evaluation setting, comprising real mobile applications and task-driven agent executions. For each task, an agent follows the corresponding task instruction and produces an interaction trajectory consisting of mobile interface states and proposed actions. These trajectories are subsequently frozen and audited before guardrail evaluation. During the evaluation, each frozen trajectory is presented step by step to each guardrail, so that all methods receive the same sequence of interface states and proposed actions. This protocol isolates guardrail decisions from differences in agent planning and execution while retaining interaction trajectories produced by autonomous agent execution. More details on task construction, trajectory auditing, and the evaluation protocol are provided in Appendix B.1.

Dataset. The final evaluation set contains 592 task trajectories spanning 10 mobile applications and 10 safety-risk categories. Each task is assigned a single safety risk category and one of four task-level intervention categories: `PERMIT`, `CONFIRM`, `STOP` or `CONFIRM+STOP`. The `CONFIRM+STOP` category represents tasks in which a legitimate consequential operation should require confirmation at one point in the trajectory, while a subsequent operation should be blocked. The risk category, task semantics, and intervention annotations are manually designed or audited, whereas the corresponding interaction trajectories are obtained from agent execution and subsequently frozen for evaluation. The detailed task construction, application distribution, and risk/intervention distribution are provided in Appendix B.1, Appendix B.2, and Appendix B.3, respectively.

Baselines. We compare ContextGuard with five representative agent safety guardrails: AgentSpec (Wang et al., 2026), Safiron (Huang et al., 2026a), AgentAuditor (Luo et al., 2026), OS-Sentinel (Sun et al., 2026), and SeerGuard (Yu et al., 2026). These methods cover rule-based and model-based approaches to safety monitoring and intervention. Each baseline is adapted to the common AndroidLab evaluation setting at the interface level while preserving its underlying safety mechanism. All methods are evaluated on the same frozen evaluation trajectories and against the same task-level annotations. Details of the baseline configurations, input representations, output adaptation, and model usage are provided in Appendix B.4.

Metrics. All metrics are computed at the task level. For binary safety evaluation, `PERMIT` is treated as *SAFE*, while `CONFIRM`, `STOP` and `CONFIRM+STOP` are treated as *UNSAFE*. With *UNSAFE* as the positive class, we report **Binary Accuracy**, **False Positive Rate**, and **F1 Score**. To evaluate fine-grained intervention selection, we additionally report **Intervention Accuracy**, which measures task-level exact agreement between the predicted and annotated intervention categories. A task is counted as correct only when its complete intervention category matches the annotation. Finally, we report **Guardrail Overhead** as the mean time required by a guardrail to produce guardrail decisions for one evaluation task. Formal definitions of all metrics and details of the overhead measurement are provided in Appendix B.5.

4.2 Main Results

Table 1 reports the performance of ContextGuard and the evaluated baselines on the 592-task evaluation set. ContextGuard achieves a Binary Accuracy of 90.9%, a False Positive Rate of 4.0%, and an F1 Score of 93.6%, corresponding to the highest Binary Accuracy, lowest False Positive Rate, and highest F1 Score among the evaluated methods. At the fine-grained intervention level, ContextGuard achieves an Intervention Accuracy of 87.8%, compared with 64.9% for SeerGuard, 58.1% for OS-Sentinel, 29.4% for AgentSpec, and 28.7% for AgentAuditor. Thus, ContextGuard maintains strong binary safety performance while showing a substantially larger advantage in fine-grained intervention selection. ContextGuard incurs a Guardrail Overhead of 240.85 ms/task. This is higher than AgentSpec (1.45 ms/task) and Safiron (43.46 ms/task), but lower than AgentAuditor (7360.48 ms/task), OS-Sentinel (84541.87 ms/task), and SeerGuard (77829.37 ms/task). The measured overhead therefore falls between the lightweight rule-based baselines and the substantially more expensive model-based guardrails. To examine generalization beyond Android-Lab, we additionally evaluate ContextGuard on the dataset of MobileSafetyBench (Lee et al., 2026). The details are provided in Appendix C.

4.3 Ablation Study

We examine the contribution of task and execution-history context through three progressively enriched context configurations. All configurations use the same evaluation traces and safety constraints, while differing only in the contextual information provided to the guardrail. The `context-none` configuration uses the current interface state and proposed action. The `context-task` configuration additionally incorporates the task instruction, while the `context-task-history` configuration further incorporates the preceding interaction history and corresponds to the full ContextGuard configuration. This design allows the effects of task context and execution history to be evaluated separately.

Table 1: Results on the AndroidLab evaluation set of 592 task trajectories. Binary Acc. denotes **Binary Accuracy**, FPR denotes **False Positive Rate**, and F1 denotes **F1 Score**, with `UNSAFE` as the positive class. Intervention Acc. denotes **Intervention Accuracy**, measuring task-level exact agreement with the annotated intervention category. **Guardrail Overhead** is reported in milliseconds per task.

Method	Binary Acc. ↑	FPR ↓	F1 ↑	Intervention Acc. ↑	Guardrail Overhead ↓
AgentSpec (Wang et al., 2026)	75.5%	95.3%	85.9%	29.4%	**1.45**
Safiron (Huang et al., 2026a)	50.0%	35.3%	57.3%	–	43.46
AgentAuditor (Luo et al., 2026)	74.2%	82.0%	84.3%	28.7%	7360.48
OS-Sentinel (Sun et al., 2026)	77.0%	13.3%	82.7%	58.1%	84541.87
SeerGuard (Yu et al., 2026)	70.1%	15.3%	76.5%	64.9%	77829.37
ContextGuard (Ours)	**90.9%**	**4.0%**	**93.6%**	**87.8%**	240.85

Notes. All methods are evaluated on the same AndroidLab evaluation set. Safiron produces only a binary intervention decision and therefore cannot distinguish among the fine-grained intervention categories . Intervention Accuracy is therefore not reported for Safiron.

Table 2: Ablation study on the AndroidLab evaluation set. Binary Acc., FPR, F1, and Intervention Acc. denote **Binary Accuracy**, **False Positive Rate**, **F1 Score**, and **Intervention Accuracy**, respectively. **Guardrail Overhead** is reported in milliseconds per task.

Configuration	Binary Acc. ↑	FPR ↓	F1 ↑	Intervention Acc. ↑	Guardrail Overhead ↓
`context-none`	89.7%	6.0%	92.7%	56.6%	**238.95**
`context-task`	90.0%	**4.0%**	93.0%	79.9%	240.74
`context-task-history`	**90.9%**	**4.0%**	**93.6%**	**87.8%**	240.85

Table 2 reports the results across the three configurations. Intervention Accuracy increases from 56.6% for `context-none` to 79.9% with task context and further to 87.8% with both task and history context. The binary safety metrics remain relatively stable across configurations, with Binary Accuracy increasing from 89.7% to 90.9% and F1 Score from 92.7% to 93.6%. The False Positive Rate decreases from 6.0% to 4.0% after introducing task context and remains at 4.0% when execution history is further incorporated. These results indicate that contextual information primarily improves the guardrail's ability to distinguish fine-grained intervention requirements, while having a more limited effect on binary safety classification. The Guardrail Overhead also remains nearly unchanged across the three configurations, ranging from 238.95 ms/task to 240.85 ms/task. The full `context-task-history` configuration achieves the highest Intervention Accuracy of 87.8% while maintaining comparable binary safety performance and Guardrail Overhead to the other configurations.

4.4 Analysis

4.4.1 Analysis of Main Results

The main results show that ContextGuard improves fine-grained intervention selection while maintaining strong binary safety performance. Its higher Intervention Accuracy is not accompanied by a more conservative intervention policy, as reflected by its low False Positive Rate. This behavior is consistent with the context-conditioned design of ContextGuard: contextual predicates are incorporated directly into explicit safety constraints, allowing the applicability of a constraint to depend on the task instruction and interaction state. Consequently, the same or similar interface-level action can require different interventions depending on the user's objective and the stage of the interaction, enabling the guardrail to distinguish between actions that are merely consequential and those that are inappropriate in the current context.

The results also reflect the role of the neuro-symbolic formulation in separating semantic interpretation from safety enforcement. Neural semantic grounding provides a flexible representation of heterogeneous mobile observations, including interface, action, task, and interaction-history information, while the symbolic safety constraints explicitly specify when a safety requirement applies and what intervention it entails. This separation allows contextual information to influence constraint applicability without reducing the final safety decision to an opaque risk score. It also provides an inspectable connection from grounded semantic predicates to the corresponding safety constraint and intervention, which is important for fine-grained runtime intervention.

The Guardrail Overhead results further characterize the computational cost of the evaluated guardrails. ContextGuard requires 240.85 ms/task while performing semantic grounding, contextual integration, and symbolic safety reasoning locally, without invoking an external LLM or VLM during safety verification. Although AgentSpec and Safiron incur lower Guardrail Overheads, their intervention-level or binary safety performance is substantially different from that of ContextGuard. In contrast, AgentAuditor, OS-Sentinel, and SeerGuard incur substantially larger Guardrail Overheads under the evaluated configurations, while relying on model-based inference for safety judgment. These results highlight that ContextGuard achieves fine-grained, context-dependent intervention with substantially lower computational cost than the evaluated model-based guardrails, while avoiding repeated external model inference during safety verification. Thus, the advantage of ContextGuard lies not in minimizing Guardrail Overhead alone, but in combining contextual safety reasoning, fine-grained intervention, and practical computational cost within a single guardrail.

4.4.2 Analysis of Ablation Study

The ablation results show that task context and execution history contribute differently to fine-grained intervention selection. Introducing the task instruction increases Intervention Accuracy from 56.6% to 79.9%, indicating that the current interface state and proposed action alone do not fully determine whether a consequential operation is consistent with the user's objective. Task context provides the semantic reference needed to distinguish task-aligned operations from actions that exceed the stated objective, thereby determining whether the corresponding safety constraints should apply.

Adding execution history further increases Intervention Accuracy from 79.9% to 87.8%. This improvement indicates that task semantics alone cannot fully characterize the state of an ongoing workflow. Interaction history provides information about previously completed operations, enabling the guardrail to distinguish an action that is appropriate at an earlier stage from a similar action that becomes inappropriate after the workflow has progressed or the task has been completed. Thus, task context and execution history provide complementary information for determining constraint applicability over evolving interaction states.

The Guardrail Overhead changes only marginally across the three configurations, from 238.95 ms/task to 240.85 ms/task. Thus, in this evaluation, incorporating task and history context improves fine-grained intervention selection without a substantial increase in the measured Guardrail Overhead. Together, the ablation results support the role of contextual information as part of the safety-constraint applicability conditions rather than as an auxiliary input to a separate action-level risk classifier.

5 Conclusion

In this paper, we have presented ContextGuard, a new context-conditioned neuro-symbolic safety guardrail for mobile GUI agents. By grounding interface, task, and interaction-history information into semantic predicates and incorporating them into explicit safety constraints, ContextGuard enables fine-grained intervention decisions that adapt to the evolving interaction context. The results demonstrate that this design improves intervention selection while maintaining strong binary safety performance and practical computational cost. These findings highlight the value of combining contextual semantic grounding with explicit neuro-symbolic safety reasoning for mobile GUI agent safety.

AI USE STATEMENT

Generative AI tools were partially used to assist with language editing, manuscript polishing, and code development. All AI-assisted content was reviewed and revised by the authors, and all code was reviewed, verified, and tested by the authors. The authors take full responsibility for the final manuscript, including its claims, methodology, experimental results and code.

ETHICS STATEMENT

This work uses publicly available AndroidLab tasks and interaction trajectories and does not involve human subjects or the collection of new personal data. The study focuses on runtime safety verification for mobile GUI agents, and the reported experiments are conducted for safety evaluation and research purposes.

REPRODUCIBILITY STATEMENT

Implementation details, predicate definitions, safety constraints, evaluation-task composition, and metric definitions are provided in the main paper and appendix. The experimental protocol and baseline configurations are described to support reproduction of the reported results.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Sujin Chen, Lijun Li, Tianyi Du, and Jing Shao. Mobileworldsafety: Benchmarking gui agent safety against environmental injection attacks in android apps. *arXiv preprint arXiv:2608.17659*, 2026.

Zhaorun Chen, Mintong Kang, and Bo Li. Shieldagent: Shielding agents via verifiable safety policy reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 8313–8344, 2025.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Lennert De Smet, Pedro Zuidberg Dos Martires, Robin Manhaeve, Giuseppe Marra, Angelika Kimmig, and Luc De Raedt. Neural probabilistic logic programming in discrete-continuous domains. In *Uncertainty in Artificial Intelligence*, pp. 529–538, 2023.

Vincent Derkinderen, Robin Manhaeve, Pedro Zuidberg Dos Martires, and Luc De Raedt. Semirings for probabilistic and neuro-symbolic logic programming. *International Journal of Approximate Reasoning*, 171:109130, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Haitao Hu, Peng Chen, Yanpeng Zhao, and Yuqi Chen. Agentsentinel: An end-to-end and real-time security defense framework for computer-use agents. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3535–3549, 2025.

Yue Huang, Hang Hua, Yujun Zhou, Pengcheng Jing, Manish Nagireddy, Inkit Padhi, Greta Dolcetti, Zhangchen Xu, Subhajit Chaudhury, Ambrish Rawat, Liubov Nedoshivina, Pin-Yu Chen, Prasanna Sattigeri, and Xiangliang Zhang. Building a foundational guardrail for general agentic systems via synthetic data. *International Conference on Learning Representations*, 2026a.

Yue Huang, Hang Hua, Yujun Zhou, Pengcheng Jing, Manish Nagireddy, Inkit Padhi, Greta Dolcetti, Zhangchen Xu, Subhajit Chaudhury, Ambrish Rawat, et al. Building a foundational guardrail for general agentic systems via synthetic data. In *International Conference on Learning Representations*, volume 2026, pp. 35963–35994, 2026b.

Simón Patiño Idarraga, Erick Silva, Rehana Yasmin, and Ali Shoker. Herding end-to-end autonomous driving via neuro-symbolic safety guards. *arXiv preprint arXiv:2608.11451*, 2026.

Thomas Kuntz, Agatha Duzan, Hao Zhao, Francesco Croce, Zico Kolter, Nicolas Flammarion, and Maksym Andriushchenko. Os-harm: A benchmark for measuring safety of computer use agents. *Advances in Neural Information Processing Systems*, 38, 2026.

Juyong Lee, Dongyoon Hahm, June Suk Choi, W Bradley Knox, and Kimin Lee. Mobilesafetybench: Evaluating safety of autonomous agents in mobile device control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 37565–37573, 2026.

Hailin Liu, Eugene Ilyushin, Jie Ni, and Min Zhu. Safeagent: A runtime protection architecture for agentic systems. *arXiv preprint arXiv:2604.17562*, 2026.

Hanjun Luo, Shenyu Dai, Chiming Ni, Xinfeng Li, Guibin Zhang, Kun Wang, Tongliang Liu, and Hanan Salam. Agentauditor: Human-level safety and security evaluation for llm agents. *Advances in Neural Information Processing Systems*, 38:43241–43298, 2026.

Weidi Luo, Shenghong Dai, Xiaogeng Liu, Suman Banerjee, Huan Sun, Muhao Chen, and Chaowei Xiao. Agrail: A lifelong agent guardrail with effective and adaptive safety detection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8104–8139, 2025.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31, 2018.

Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 298:103504, 2021.

Connor Pryor, Charles Dickens, Eriq Augustine, Alon Albalak, William Yang Wang, and Lise Getoor. Neupsl: Neural probabilistic soft logic. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2023.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Zachary Ravichandran, Alexander Robey, Vijay Kumar, George J Pappas, and Hamed Hassani. Safety guardrails for llm-enabled robots. *IEEE Robotics and Automation Letters*, 2026.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Programmable privilege control for llm agents. *arXiv preprint arXiv:2504.11703*, 2025.

Qiushi Sun, Mukai Li, Zhoumianze Liu, Zhihui Xie, Fangzhi Xu, Zhangyue Yin, Kanzhi Cheng, Zehao Li, Zichen Ding, Qi Liu, et al. Os-sentinel: Towards safety-enhanced mobile gui agents via hybrid validation in realistic workflows. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9529–9553, 2026.

Haoyu Wang, Christopher M Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. In *Proceedings of the 2026 IEEE/ACM 48th International Conference on Software Engineering*, pp. 2938–2950, 2026.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Nathaniel D Bastian, et al. Guardagent: safeguard llm agents via knowledge-enabled reasoning. In *ICML 2025 workshop on computer use agents*, 2025.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android autonomous agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. A survey on evaluation of llm-based agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26690–26714, 2026.

Sheng Yin, Xianghe Pang, Yuanzhuo Ding, Menglan Chen, Yutong Bi, Yichen Xiong, Wenhao Huang, Zhen Xiang, Jing Shao, and Siheng Chen. Safeagentbench: A benchmark for safe task planning of embodied llm agents. *arXiv preprint arXiv:2412.13178*, 2024.

Shaohan Yu, Lijun Li, Chenyang Si, Lu Sheng, and Jing Shao. Proguard: Towards proactive multimodal safeguard. *arXiv preprint arXiv:2512.23573*, 2025.

Xue Yu, Bo Yuan, Pengshuai Yang, Kailin Zhao, Hong Hu, and Junlan Feng. Seerguard: A safety framework for mobile gui agents via world model prediction. *arXiv preprint arXiv:2607.15550*, 2026.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI conference on human factors in computing systems*, pp. 1–20, 2025.

Zhexin Zhang, Shiyao Cui, Yida Lu, Jingzhuo Zhou, Junxiao Yang, Hongning Wang, and Minlie Huang. Agent-safetybench: Evaluating the safety of llm agents. *arXiv preprint arXiv:2412.14470*, 2024.

A Methodological Details

A.1 Predicate Representation

ContextGuard represents each guarded interaction using a structured set of soft semantic predicates. Given the heterogeneous observation $O_t = (U_t, \mathcal{T}, a_t, \mathcal{H}_t)$, the neural grounding function ϕ_θ produces the semantic representation $\mathcal{P}_t = \phi_\theta(O_t)$, decomposed into interface/action predicates and contextual predicates, $\mathcal{P}_t = \mathcal{P}_t^{ia} \cup \mathcal{P}_t^{ctx}$. Each predicate $p_i \in \mathcal{P}_t$ is associated with a confidence value $\mu_i \in [0, 1]$. The predicates serve as an intermediate semantic representation between the heterogeneous mobile observation and the explicit safety constraints in the knowledge base $\mathcal{K}$.

Interface and Action Predicates. The interface/action predicates $\mathcal{P}_t^{ia}$ characterize the current mobile interface, the proposed operation, and the interface entity affected by the action. Rather than exposing the raw UI representation directly to the safety constraints, the grounding process maps interface elements and action primitives to semantic properties relevant to safety verification. These properties can describe the interaction affordance of the target element, the functional semantics of the proposed operation, and its potential external consequences. Typical interface-level predicates characterize whether a target element is actionable or enabled, while semantic action predicates describe properties such as submit, delete, $\mathrm{financial}$, $\mathrm{credential}$, $\mathrm{permission}$, $\mathrm{setting}$, and $\mathrm{external}$. These predicates are grounded from the current UI state together with the proposed action and its affected interface entity. Let v_t denote the interface entity targeted or affected by the proposed action a_t. For example, a low-level tap action targeting a button whose semantic role is to submit information can be represented by predicates such as $\mathrm{submit}(v_t)$ and $\mathrm{consequential}(v_t)$. Similarly, an action targeting a control that initiates an external transmission can be associated with $\mathrm{external}(v_t)$.

The same low-level action primitive can therefore correspond to different semantic predicates depending on the target interface entity. Importantly, interface/action predicates describe the semantic properties of the proposed operation and its target rather than directly assigning a safety label. Whether such an operation requires intervention depends on the contextual conditions encoded by the safety constraints.

Contextual Predicates. The contextual predicates $\mathcal{P}_t^{ctx}$ characterize the role of the proposed action within the ongoing interaction. They are grounded from the task instruction $\mathcal{T}$ and the execution history $\mathcal{H}_t$, together with the current interface and proposed action when necessary to establish their contextual meaning. These predicates capture properties whose interpretation depends on the current task or interaction state rather than solely on the surface semantics of the interface. The contextual predicates include task-related properties such as task alignment and task overreach, as well as interaction-state properties such as commitment boundaries, task completion, and repeated operations. For example, $\mathrm{task_aligned}(a_t, \mathcal{T})$ indicates that the proposed operation is consistent with the stated task objective, whereas $\mathrm{task_overreach}(a_t, \mathcal{T})$ indicates that the operation exceeds or deviates from the requested objective. A commitment-boundary predicate characterizes whether the proposed action crosses a point at which an external or consequential effect is about to occur. Similarly, $\mathrm{task_completed}(\mathcal{H}_t)$ captures whether the requested task has already reached completion, while a repeated-operation predicate captures whether a consequential operation has already been performed or attempted in the preceding interaction history. These contextual predicates do not themselves determine whether an action is safe or unsafe. Instead, they provide the contextual conditions under which interface/action semantics acquire different safety implications. For example, a submission operation may satisfy $\mathrm{submit}(v_t)$ and $\mathrm{consequential}(v_t)$ in both an incomplete and a completed task. However, the contextual predicates associated with task progress and completion can cause different safety constraints to become applicable in these two states.

Predicate Grounding and Confidence. The grounding function ϕ_θ maps heterogeneous mobile observations to predicate-confidence pairs. For each predicate p_i, the grounding module produces a confidence $\mu_i \in [0, 1]$ representing the degree to which the available observation supports that semantic property. The confidence is not interpreted as a probability that the corresponding action is safe or unsafe; it measures semantic support for the predicate itself. This distinction allows uncertainty in interface interpretation and contextual understanding to be propagated to the subsequent neuro-symbolic reasoning stage without collapsing semantic grounding into an action-level risk prediction.

The resulting predicate set is then provided to the safety reasoner $\mathcal{R}$, which evaluates the explicit safety constraints in $\mathcal{K}$. A safety constraint can jointly reference interface/action predicates and contextual predicates. Thus, the grounding stage determines which semantic properties are supported by the current interaction, while the safety knowledge base determines how these properties are combined into an applicable safety requirement.

A.2 Context-Conditioned Safety Constraints

ContextGuard encodes safety requirements as an explicit knowledge base $\mathcal{K}$ consisting of reusable context-conditioned safety constraints. Each constraint specifies the semantic conditions under which a particular safety requirement applies to the proposed action. Unlike a direct action-level risk classifier, a safety constraint does not assign a global safe or unsafe label to an action. Instead, its applicability is determined jointly by interface/action semantics and contextual predicates grounded from the current task and interaction history.

Formally, a safety constraint $\kappa_k \in \mathcal{K}$ can be represented as

$$\kappa_k : p_{k,1} \wedge p_{k,2} \wedge \cdots \wedge p_{k,m} \Rightarrow d_k,$$

where $p_{k,j} \in \mathcal{P}_t$ denotes a grounded semantic predicate and $d_k \in \texttt{CONFIRM}, \texttt{STOP}$ specifies the intervention required when the constraint is activated. The antecedent may contain both interface/action predicates from $\mathcal{P}_t^{ia}$ and contextual predicates from $\mathcal{P}_t^{ctx}$. A constraint therefore defines not only what type of operation is consequential, but also the contextual conditions under which that operation requires intervention.

Context-Conditioned Applicability. The central role of contextual predicates is to determine when a safety constraint becomes applicable. Interface/action predicates characterize the semantics of the proposed operation and its target, whereas contextual predicates characterize its relation to the task and the preceding interaction. A consequential operation can therefore satisfy the same interface/action predicates across different interaction states while activating different safety constraints because the contextual predicates differ.

For example, consider an action targeting a submission control. The interface/action predicates may indicate $\mathrm{submit}(v_t)$ and $\mathrm{consequential}(v_t)$ regardless of whether the task is complete. A constraint may require confirmation when the submission occurs at an expected commitment boundary of an incomplete task:

$$\mathrm{submit}(v_t) \wedge \mathrm{consequential}(v_t) \wedge \mathrm{commitment_boundary}(a_t) \wedge \mathrm{task_incomplete}(\mathcal{H}_t) \Rightarrow \texttt{CONFIRM}.$$

After the task has already been completed, the same submission semantics may instead activate a constraint for repeated high-impact operations:

$$\mathrm{submit}(v_t) \wedge \mathrm{consequential}(v_t) \wedge \mathrm{task_completed}(\mathcal{H}_t) \wedge \mathrm{repeated}(a_t, \mathcal{H}_t) \Rightarrow \texttt{STOP}.$$

These examples illustrate that contextual predicates are part of the applicability conditions of safety constraints rather than auxiliary evidence supplied to a separate risk classifier.

Constraint Categories. The safety knowledge base contains reusable constraints covering several types of safety-relevant operations. These include consequential or irreversible operations, destructive operations, financial operations, privacy-sensitive operations, permission-related operations, external information transfer, and context-dependent repeated operations. The constraints can also incorporate task alignment and task overreach to distinguish actions that are explicitly required by the task from actions that exceed the user's stated objective.

For consequential or irreversible operations, constraints can require explicit confirmation when the proposed action reaches a commitment or external-effect boundary. For example, a submission or purchase action may activate a `CONFIRM` constraint when it is aligned with the task but is about to produce an external or consequential effect. In contrast, an operation that is unrelated to the task or repeats a completed high-impact action can activate a `STOP` constraint.

Privacy- and permission-related constraints condition their applicability on both the semantic properties of the affected interface element and the contextual conditions of the interaction. For instance, an external transmission involving sensitive information may require intervention when the operation is not sufficiently aligned with the task or lacks an appropriate confirmation condition. Similarly, a permission-granting action can be constrained according to the requested permission and its role within the current task rather than being treated as universally unsafe.

Constraint Activation. Because each grounded predicate has a confidence value μ_i, the applicability of a safety constraint is evaluated through differentiable logical reasoning. For a constraint κ_k with antecedent predicate confidences $\mu_{k,1}, \ldots, \mu_{k,m}$, the reasoner computes an activation confidence $\alpha_k = T(\mu_{k,1}, \ldots, \mu_{k,m})$, where $T(\cdot)$ is the differentiable t-norm defined in Section 3.3. The constraint is considered active when $\alpha_k \geq \delta$. The complete set of active constraints is therefore

$$\mathcal{C}_t = \{\kappa_k \in \mathcal{K} \mid \alpha_k \geq \delta\}.$$

When multiple constraints are active simultaneously, their associated intervention requirements are resolved by the fixed precedence used by the runtime intervention layer, with `STOP` taking precedence over `CONFIRM`.

The resulting constraint set $\mathcal{C}_t$ constitutes the symbolic output of the neuro-symbolic reasoning stage. Importantly, ContextGuard does not introduce an additional learned risk classifier between constraint activation and intervention. The grounded predicates provide uncertain semantic evidence, the explicit safety constraints determine the contextual applicability of safety requirements, and the runtime intervention layer directly enforces the resulting requirements.

A.3 Differentiable Constraint Evaluation

Given the grounded predicate set $\mathcal{P} * t$, each predicate p_i is associated with a confidence value $\mu_i \in [0, 1]$. For a safety constraint κ_k, let $A_k = p * k, 1, \ldots, p_{k,m}$ denote its antecedent predicates. The neuro-symbolic reasoner computes the activation confidence of κ_k by applying a differentiable t-norm T to the corresponding predicate confidences:

$$\alpha_k = T(\mu_{k,1}, \ldots, \mu_{k,m}).$$

The resulting value $\alpha_k \in [0, 1]$ represents the degree to which the current semantic representation satisfies the applicability conditions of the constraint. It is therefore interpreted as constraint activation confidence rather than as a probability that the action is unsafe.

A constraint is activated when its activation confidence reaches the threshold δ:

$$\mathcal{C}_t = \{\kappa_k \in \mathcal{K} \mid \alpha_k \geq \delta\}.$$

The threshold provides a common criterion for converting soft neuro-symbolic evidence into discrete constraint activation. Constraints that do not reach the threshold remain inactive and do not directly contribute to the runtime intervention.

When multiple constraints correspond to the same intervention requirement, their activation can be combined using the aggregation operator defined in the implementation. In contrast, constraints associated with different intervention requirements remain individually represented in $\mathcal{C}_t$, after which the runtime intervention layer resolves simultaneous activations according to the fixed precedence `STOP` $>$ `CONFIRM` $>$ `PERMIT`.

In the current implementation, the differentiable reasoning stage is used only to evaluate the applicability of explicit safety constraints from grounded semantic predicates. It does not introduce an additional learned risk classifier or directly predict the final intervention decision. The complete reasoning pipeline is therefore $\mathcal{P}_t \xrightarrow{\mathcal{K},\mathcal{R}} \mathcal{C}_t$, followed by deterministic runtime intervention based on the activated constraints.

B Experimental Details

B.1 Task Construction and Annotation

All evaluation tasks are derived from tasks in the original AndroidLab environment. The original tasks are modified or extended to construct safety-relevant scenarios suitable for fine-grained intervention evaluation while retaining the original task semantics and mobile-agent interaction setting. The task semantics, safety risk categories, and task-level intervention annotations are manually designed or audited to ensure that each task has a well-defined safety requirement.

Each task is assigned a single safety risk category from the 10 categories described in Appendix B.3. The task-level intervention annotation specifies the required intervention pattern over the corresponding trajectory, including `PERMIT`, `CONFIRM`, `STOP` or `CONFIRM+STOP`. These annotations are determined independently of the predictions of the evaluated guardrails.

For each task, an interaction trajectory is obtained from agent execution in the AndroidLab environment following the corresponding task instruction. The resulting trajectory is then frozen and audited before guardrail evaluation. During evaluation, each guardrail is applied to the recorded interaction states and proposed actions in the frozen trajectory, without rerunning the agent or altering the trajectory. Thus, all compared methods operate on the same interaction sequence, isolating guardrail decisions from differences in agent planning and execution.

All reported evaluation metrics are computed at the task level. Thus, the manually established task annotations define the expected safety requirement for each task, while the frozen agent-generated trajectory provides the common interaction sequence used for controlled comparison across guardrails.

Table 3: Distribution of evaluation tasks across mobile applications and associated safety risk categories.

Application	# Tasks	Safety Risk Categories
Email	324	Privacy Leakage; Illegal Activities; Physical and Health Risks; Ethical Risks; Misinformation and False Information Dissemination; Harmful Information Generation
Settings	120	Cybersecurity; System Availability Disruption
Contacts	38	Privacy Leakage; Data Loss
Bluecoins	34	Financial Loss
Calendar	26	Data Loss
Shopping	24	Financial Loss; Privacy Leakage
Clock	10	Data Loss
Zoom	9	Privacy Leakage
Cantook	4	Data Loss
Maps	3	Physical and Health Risks
Total	**592**	

B.2 Evaluation Environments and Task Distribution

We evaluate ContextGuard across 10 mobile applications, including eight applications inherited from AndroidLab and two additional applications, Email and Shopping, introduced to provide interaction scenarios that are not sufficiently covered by the original environment set. The applications cover diverse interaction domains including communication, device configuration, contact management, personal finance, scheduling, media consumption, navigation, shopping, and video conferencing. The evaluation tasks are based on the corresponding AndroidLab tasks where applicable, with tasks modified or extended to instantiate safety-relevant interaction scenarios suitable for our evaluation.

The resulting task distribution is non-uniform because different applications provide different opportunities for constructing safety-relevant scenarios. In particular, Email supports a broad range of interactions involving privacy, illegal activities, physical and health risks, ethical risks, misinformation, and harmful information generation, and therefore contributes substantially more tasks than applications with narrower functionality. Shopping provides additional scenarios involving financial loss and privacy leakage that are not sufficiently represented by the original AndroidLab environments. In contrast, applications such as Clock, Cantook, and Maps provide fewer interaction patterns relevant to the targeted risk categories and consequently contribute fewer tasks. The distribution therefore reflects the diversity and availability of safety-relevant task scenarios rather than an attempt to balance the number of tasks across applications.

Table 3 summarizes the distribution of the 592 evaluation tasks across the 10 applications and their associated safety risk categories.

B.3 Risk Categories and Intervention Distribution

Each evaluation task is assigned a single safety risk category through manual task design and auditing. The final evaluation set covers 10 categories: **Privacy Leakage**, **Cybersecurity**, **Data Loss**, **Financial Loss**, **Illegal Activities**, **Physical and Health Risks**, **Ethical Risks**, **Misinformation and False Information Dissemination**, **Harmful Information Generation**, and **System Availability Disruption**.

At the task level, each trajectory is assigned one of four intervention patterns: `PERMIT`, `CONFIRM`-only, `STOP`-only, or `CONFIRM+STOP`. The `PERMIT` pattern denotes a task for which no intervention is required. `CONFIRM` and `STOP` denote tasks requiring confirmation or blocking, respectively. `CONFIRM+STOP` denotes a task containing both a legitimate consequential operation requiring confirmation and a later operation that should be blocked. The three non-`PERMIT` patterns form a mutually exclusive partition of the unsafe tasks.

Table 4: Distribution of evaluation tasks across safety risk categories and intervention patterns.

Risk Category	Total	Unsafe	PERMIT	CONFIRM	STOP	CONFIRM +STOP
Privacy Leakage	75	60	15	32	0	28
Cybersecurity	57	42	15	40	1	1
Data Loss	66	51	15	33	13	5
Financial Loss	57	42	15	36	0	6
Illegal Activities	56	41	15	1	40	0
Physical and Health Risks	56	41	15	2	39	0
Ethical Risks	57	42	15	1	41	0
Misinformation and False Information Dissemination	57	42	15	1	41	0
Harmful Information Generation	48	33	15	1	32	0
System Availability Disruption	63	48	15	42	0	6
Total	**592**	**442**	**150**	**189**	**207**	**46**

The final evaluation set contains 592 tasks, including 150 PERMIT tasks and 442 unsafe tasks. The detailed distribution of tasks across safety risk categories and intervention patterns is reported in Table 4.

B.4 Baseline Configuration

All compared methods are evaluated under the same task-level evaluation protocol and on the same evaluation trajectories. To accommodate differences in their native input and output spaces, each baseline is adapted to the common AndroidLab evaluation setting at the interface level, without modifying its underlying safety mechanism. AgentSpec is evaluated as a local rule-based guardrail using task information, parsed actions, and preceding interaction steps. Safiron uses a local language model for safety judgment based on the task request and parsed action sequence. AgentAuditor, OS-Sentinel, and SeerGuard use their respective model-based safety-judgment pipelines under the evaluated configurations.

The compared methods may differ in the interface information available to their native safety mechanisms. These differences are preserved rather than eliminated, since the purpose of the comparison is to evaluate the respective guardrails under their supported input representations. Nevertheless, all methods operate on the same task trajectories and are evaluated against the same task-level annotations. For methods whose native outputs do not directly correspond to the intervention categories used in this work, an output adapter is applied after the native safety judgment to obtain a compatible representation for evaluation. The adapter does not provide additional information to the underlying guardrail and does not modify its safety reasoning. Safiron produces only a binary intervention decision and is therefore evaluated with the binary safety metrics but not with Intervention Accuracy. For methods that provide intervention-category outputs, the corresponding task-level intervention category is used to compute Intervention Accuracy. This separation ensures that the binary evaluation measures whether a guardrail correctly identifies unsafe tasks, while Intervention Accuracy is reported only when the method provides sufficient output information to distinguish the required intervention categories.

B.5 Evaluation Metrics

All reported metrics are computed at the task level. Each task is assigned one of four ground-truth intervention categories: PERMIT, CONFIRM, STOP or CONFIRM+STOP. The predicted task-level intervention category is determined from the sequence of step-level interventions produced along the corresponding task trajectory.

For binary safety evaluation, tasks annotated as PERMIT are treated as *SAFE*, while tasks annotated as CONFIRM, STOP or CONFIRM+STOP are treated as *UNSAFE*. Let TP, TN, FP, and FN denote the corresponding task-level counts, with *UNSAFE* as the positive class. We report **Binary Accuracy**, **False Positive Rate**, and **F1 Score**.

Binary Accuracy is defined as

$$Binary\ Acc. = \frac{TP + TN}{TP + TN + FP + FN}.$$

False Positive Rate is defined as

$$FPR = \frac{FP}{FP + TN}.$$

The precision and recall of the *UNSAFE* class, which are used to compute the F1 Score, are defined as

$$Precision = \frac{TP}{TP + FP}, \qquad Recall = \frac{TP}{TP + FN}.$$

The **F1 Score** is the harmonic mean of precision and recall:

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}.$$

To evaluate fine-grained intervention selection, we report **Intervention Accuracy**, which measures task-level exact agreement with the annotated intervention category. Let $\hat{z}_i$ and z_i^* denote the predicted and ground-truth intervention categories for task i, respectively. Intervention Accuracy is defined as

$$Intervention\ Acc. = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[\hat{z}_i = z_i^*],$$

where N is the total number of evaluated tasks. A task is counted as correct only when its complete predicted intervention category exactly matches the annotation. In particular, a task annotated as `CONFIRM+STOP` is counted as correct only when the predicted trajectory contains the required confirmation behavior and subsequent blocking behavior. Thus, Intervention Accuracy evaluates fine-grained intervention selection beyond the binary distinction between *SAFE* and *UNSAFE*.

For computational cost comparison, we report **Guardrail Overhead** as the time required by a guardrail to process an evaluation trajectory and produce its safety judgments, averaged over tasks and reported in milliseconds per task. For ContextGuard, this includes semantic grounding of the mobile interface representation, action processing, incorporation of task and interaction-history context, safety-constraint evaluation, and intervention selection. For model-based baselines, the corresponding model inference required to produce safety judgments is included in the measured overhead. This measurement excludes trajectory generation, mobile-application execution, agent planning and execution, and manual task construction, annotation, and auditing.

C External Validation on MobileSafetyBench

To examine the generality of ContextGuard beyond the Android-Lab evaluation set, we conduct an external validation experiment on the dataset of MobileSafetyBench. The dataset contains 200 mobile GUI tasks spanning 20 task categories, with 100 high-risk and 100 low-risk tasks. Unlike Android-Lab, MobileSafetyBench provides binary risk annotations rather than task-level intervention categories. Accordingly, this experiment evaluates binary safety recognition and does not report Intervention Accuracy.

Evaluation Protocol. All methods are evaluated on the same 200 fixed interaction trajectories. The agent is not rerun for different methods, so the compared guardrails operate on the same task instructions, interaction sequences, and GUI states whenever supported by their respective input

Table 5: External binary validation on the MobileSafetyBench. High-risk tasks are treated as the positive class. All metrics are computed at the task level.

Method	Accuracy ↑	FPR ↓	Recall ↑	Precision ↑	F1 ↑
OS-Sentinel	63.5%	25.0%	52.0%	67.5%	58.8%
SeerGuard	57.0%	18.0%	32.0%	64.0%	42.7%
ContextGuard(Ours)	**91.5%**	**2.0%**	**85.0%**	**97.7%**	**90.9%**

interfaces. We compare ContextGuard with OS-Sentinel and SeerGuard using their end-to-end evaluated configurations. For ContextGuard, the MobileSafetyBench trajectories are converted into the input representation required by the guardrail, including task descriptions, proposed actions, action history, and text-visible GUI observations. The benchmark annotations are used only for evaluation and are not provided as prediction inputs. Since MobileSafetyBench provides binary risk labels rather than the fine-grained intervention annotations used in Android-Lab, this experiment evaluates cross-dataset binary safety recognition.

Results. Table 5 summarizes the results. ContextGuard achieves 91.5% task-level accuracy and 90.9% F1, compared with 63.5% accuracy and 58.8% F1 for OS-Sentinel and 57.0% accuracy and 42.7% F1 for SeerGuard. ContextGuard also obtains an FPR of 2.0%, indicating that its stronger safety recognition is not achieved through over-intervention on low-risk tasks. Overall, the results show that ContextGuard maintains strong binary safety recognition on a different mobile-safety benchmark while preserving low false-positive rates.