# SAFEFORGE: DYNAMIC NEURO-SYMBOLIC SAFETY RULE SYNTHESIS FOR EMBODIED ACTION INTERCEPTION

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Embodied AI agents operating in physical environments must evaluate the safety of proposed actions before execution — a challenge that demands both formal logical guarantees and the flexibility to handle diverse object combinations unseen during design. Existing approaches rely either on rigid rule-based systems that fail on novel entity configurations, or on neural models that lack interpretability and formal safety certificates. We present **SafeForge**, a dynamic interpretable neuro-symbolic safety guardrail framework for embodied action safety verification. SafeForge operates through three stages. First, a real-time scene graph builder dynamically models the embodied interaction environment, grounding 3D object positions and semantic attributes into soft spatial predicates via differentiable sigmoid distance kernels. Next, a differentiable neuro-symbolic reasoner performs hierarchical forward chaining with learnable rule weights, instantiating high-level safety rules into scene-grounded, concrete safety conditions tailored to the observed entity configuration. Finally, an action safety judge synthesizes the proposed action type with the entities involved to produce a binary safety decision, accompanied by a full proof trace that provides human-interpretable explanations. We evaluate SafeForge on a curated benchmark of 750 tasks spanning 120 procedurally generated scenes and 11 hazard categories, with an average of 5 action steps per sequence. Extensive experimental results demonstrate that SafeForge achieves strong detection performance while maintaining differentiability for downstream policy learning, outperforming purely neural and purely symbolic baselines in safety detection performance. The source code is available at `https://anonymous.4open.science/r/SafeForge`.

## 1 INTRODUCTION

Embodied AI agents operating in physical environments must evaluate the safety of proposed actions before execution. As large language models (LLMs) (Guo et al., 2025; Achiam et al., 2023; Bai et al., 2023; Comanici et al., 2025; Yehudai et al., 2026) increasingly empower agents to perform complex tasks autonomously (Chen et al., 2023; Driess et al., 2023; Zitkovich et al., 2023; Wang et al., 2024; Huang et al., 2023), including multi-step manipulation, human–robot collaboration, and long-horizon planning (Driess et al., 2023; Liu et al., 2025), the potential for unsafe actions grows correspondingly. Inappropriate decisions—such as placing flammable objects near heat sources or handing sharp tools to children—can lead to hazardous consequences in the physical world (Xu et al., 2022; Yin et al., 2024; Zhang et al., 2024b). Ensuring the safety of agent behaviors in such environments has therefore become a critical challenge.

Dynamic safety guardrail mechanisms (Chu et al., 2024; Thaker et al., 2024) address this challenge by introducing a verification layer between action planning and execution, intercepting potentially unsafe behaviors without modifying the underlying agent model. Existing approaches, however, fall into two camps with complementary weaknesses. Rule-based methods (Wang et al., 2026; Shi et al., 2025; Chen et al., 2025) offer interpretability and explicit safety constraints, but they typically rely on manually designed or LLM-generated rules that are not sufficiently grounded in the current environmental context. For example, a high-level rule such as keeping flammable objects away from heat sources must be instantiated with respect to the concrete objects, spatial relations, and action



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.

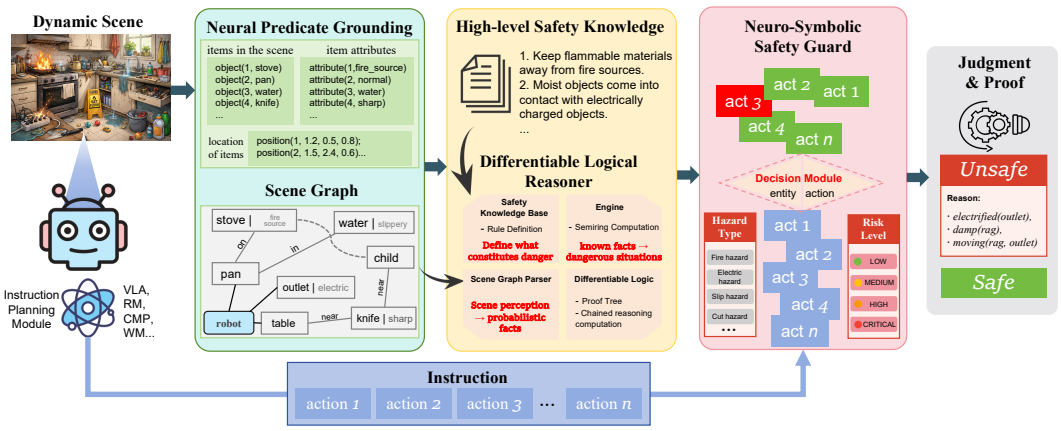


Figure 1: **Overview of SafeForge.** Given dynamic observations, SafeForge builds a probabilistic scene graph via neural predicate grounding, then synthesizes environment-aware safety rules through differentiable logical reasoning over high-level knowledge. At runtime, planned actions are checked against activated rules, and unsafe actions are intercepted with interpretable proof traces.

arguments present in a particular scene. Manually enumerating such object-specific rule variants is difficult to scale in diverse embodied environments, where agents may encounter diverse object configurations, uncertain perception, and dynamic state changes. Neural approaches (Yin et al., 2024; Yu et al., 2025; Huang et al., 2026; Xiang et al., 2025; Luo et al., 2025; Hu et al., 2025) provide flexibility through learned safety evaluation, yet often operate as black-box predictors whose decisions are difficult to inspect, calibrate, or convert into executable runtime constraints. Therefore, **neither paradigm fully bridges the gap between abstract safety knowledge and the concrete, environment-dependent constraints required for reliable action-level monitoring.**

**To address this limitation, we propose SafeForge, an environment-aware neuro-symbolic framework for dynamic safety rule construction and embodied action interception, as illustrated in Figure 1. The key idea is to separate high-level safety knowledge from its scene-specific instantiation: instead of treating safety rules as fixed constraints, SafeForge grounds reusable safety templates into executable rule instances conditioned on the current object configuration, spatial relations, and planned actions.** Our approach operates through a three-stage pipeline. First, SafeForge constructs a neuro-symbolic scene graph (Chang et al., 2023) from the observed environment, representing objects, semantic attributes, and spatial relations as probabilistic predicates. In particular, semantic properties such as `flammable` or `liquid` are grounded from object descriptors, while spatial relations such as `near` or `touching` are modeled with differentiable distance-based kernels. Second, a differentiable logical reasoner performs forward-chaining inference over a structured rule hierarchy using semiring semantics, combining grounded environmental facts with high-level safety knowledge (Manhaeve et al., 2018; De Smet et al., 2023) to derive scene-specific hazard conditions with learnable rule weights. Third, a runtime safety guardrail activates concrete rules according to their inferred confidence and matches them against the agent's planned actions. If the proposed action type and involved entities conflict with any activated rule, the action is intercepted and accompanied by a human-interpretable proof trace. In this design, the predicate grounding and probabilistic reasoning stages remain differentiable for rule confidence calibration, while the final runtime guard produces a discrete and interpretable safety decision.

**In summary, the main contributions of this paper are as follows:**

- We propose a novel environment-aware framework for constructing safety rules for embodied agents, which automatically derives concrete safety constraints from environmental context and high-level safety knowledge, reducing the reliance on manually enumerate safety rules, learned safety models or purely LLM-based judgments.
- We have designed a novel differentiable neural-symbolic reasoning mechanism that bridges perceptual uncertainty and symbolic safe logic: differentiable spatial kernels generate soft-valued predicates from raw geometric observations, while semiring-based forward chaining with learnable

rule weights enables end-to-end optimization of safe rule activation. SafeForge innovatively places both perception and reasoning within a unified probabilistic framework.

- We conduct comprehensive experiments on SafeAgentBench (Yin et al., 2024) across multiple task types, including detailed, abstract, and long-horizon tasks. The results demonstrate that our method effectively mitigates the safety–utility trade-off and exhibits strong generalization across varying levels of task complexity.

## 2 Related Work

### 2.1 Embodied Agent Safety

Recent advances in large language models (LLMs) (Guo et al., 2025; Achiam et al., 2023; Bai et al., 2023; Comanici et al., 2025; Yehudai et al., 2026), vision language models (VLMs) (Zhang et al., 2024a), and vision language action models (VLAs) (Ma et al., 2026) have significantly expanded the capabilities of embodied agents operating in interactive environments (Chen et al., 2023; Driess et al., 2023; Zitkovich et al., 2023; Wang et al., 2024; Huang et al., 2023). LLMs provide strong language understanding and reasoning capabilities, enabling agents to interpret natural language instructions and perform high-level task planning. VLMs further enhance environmental understanding by recognizing objects, attributes, and relationships from visual observations and aligning them with language-based task descriptions. Building on these capabilities, VLAs integrate perception and action generation within a unified framework, enabling agents to directly map multimodal observations to executable actions.

As embodied agents increasingly interact with physical environments, their actions introduce safety risks that differ from those in purely software-based systems (Wang et al., 2025; Perlo et al., 2025). Incorrect environmental understanding, improper device operation, or errors accumulated during long-horizon execution may lead to hazardous outcomes. In human–agent collaboration, inappropriate actions may also affect user safety or disrupt task execution. Several studies (Xu et al., 2022; Yin et al., 2024; Zhang et al., 2024b) have therefore investigated hazardous situations in embodied environments and developed task environments or benchmarks for evaluating agent safety. These efforts provide useful insights into embodied-agent failure modes and motivate the development of runtime safety mechanisms.

### 2.2 Runtime Safety Guardrail Mechanisms

To mitigate potential risks arising from agent behaviors, several studies (Chu et al., 2024; Thaker et al., 2024; Wang et al., 2026; Shi et al., 2025; Chen et al., 2025; Yin et al., 2024; Yu et al., 2025; Huang et al., 2026; Xiang et al., 2025; Luo et al., 2025; Hu et al., 2025; Lu et al., 2025) have introduced runtime safety guardrail mechanisms. These approaches typically operate between action planning and execution, evaluating generated action plans and intervening when potentially unsafe behaviors are detected. In the formal verification community, runtime enforcement has also been studied through shielding, where external mechanisms monitor agent or controller actions against a given safety specification (Bloem et al., 2015; Alshiekh et al., 2018; Carr et al., 2023; Banerjee et al., 2024; Teuber et al., 2024; Rodríguez et al., 2025). Recent extensions have considered partial observability, dynamic environments, verification-guided intervention, and richer formal specifications.

Rule-based and constraint-based approaches (Wang et al., 2026; Shi et al., 2025; Chen et al., 2025) provide explicit mechanisms for verifying agent actions. In many such systems, safety constraints are manually designed based on domain knowledge to restrict unsafe operations under specific environmental conditions. However, manually designed rules require substantial domain expertise and engineering effort, and may become difficult to maintain as environments contain diverse objects and dynamic relationships. Recent studies (Wang et al., 2026) have explored leveraging LLMs to automatically generate safety rules from natural-language safety guidelines, reducing the burden of manual rule engineering. Nevertheless, rules derived primarily from textual knowledge may lack grounding in the current environmental context, limiting their effectiveness when safety conditions depend on specific objects or spatial relationships.

Learning-based approaches (Yin et al., 2024; Yu et al., 2025; Huang et al., 2026; Xiang et al., 2025; Luo et al., 2025; Hu et al., 2025; Lu et al., 2025) instead learn safety judgments from data. Some

approaches fine-tune LLMs as safety critics that review agent-generated plans and determine whether they violate safety requirements (Xiang et al., 2025; Luo et al., 2025; Hu et al., 2025). Other methods train dedicated safety models using datasets of risky or unsafe behaviors (Yin et al., 2024; Yu et al., 2025; Huang et al., 2026), enabling the detection of potentially hazardous actions or prediction of unsafe outcomes before execution. These approaches provide flexible safety assessments, but their decisions may be less interpretable than explicit rule-based verification, and adapting them to new environments can require additional training data.

Across these approaches, a key challenge is obtaining safety constraints that accurately capture the risks associated with the current environment. In embodied settings, safety conditions may depend on environmental states, object properties, spatial relationships, and interactions among multiple objects, while high-level safety knowledge is often expressed in more abstract form. Directly translating such knowledge into executable constraints therefore remains challenging. SafeForge addresses this problem by combining environmental facts with high-level safety knowledge to construct concrete, environment-aware safety constraints for planned-action verification.

# 3 METHODOLOGY

As illustrated in Figure 1, SafeForge operates through two complementary phases: *safety rules construction* and *runtime safety guard*. Both phases share a common three-stage pipeline consisting of (1) scene dynamic detection and scene graph construction, (2) differentiable logical reasoning, and (3) neuro-symbolic safety guardrailing. Given high-level safety knowledge and the current environmental state, SafeForge grounds relevant predicates from the observed scene and instantiates the corresponding safety rules into concrete constraints for planned actions. These constraints are subsequently evaluated by the runtime guard before action execution.

## 3.1 SCENE DYNAMIC DETECTION AND SCENE GRAPH CONSTRUCTION

To enable structured safety reasoning over embodied environments, SafeForge constructs a neuro-symbolic scene graph from structured environmental observations provided by a perception module. This stage transforms unstructured sensor data into a probabilistic symbolic representation that preserves both semantic and spatial information while maintaining differentiability for downstream learning.

**Scene Representation.** Let the environment observation be denoted by $E$. SafeForge first extracts a structured scene representation

$$G = (E_o, E_r, E_a), \tag{1}$$

where $E_o = \{o_1, \ldots, o_N\}$ denotes the set of detected object instances, $E_r$ denotes the set of relational observations between objects, and $E_a$ denotes the set of object attributes and states. Each object $o_i$ is characterized by a semantic descriptor $s_i$, an attribute vector $a_i$, and a 3D position $x_i \in \mathbb{R}^3$. In our implementation, $G$ is obtained by parsing structured predicate outputs from a vision-language perception module (e.g., LLaVA-3D) that produces Prolog-style symbolic annotations of the observed scene. The perception module serves as a pluggable grounding component; SafeForge operates on the resulting object attributes, relations, and confidence values rather than relying on a specific perception architecture.

**Neural Predicate Grounding.** Based on the scene representation, SafeForge constructs a probabilistic fact set

$$F = \{(p_i, \Pr(p_i)) \mid p_i \in \mathcal{G}(G)\}, \tag{2}$$

where each $p_i$ is a grounded symbolic predicate and $\Pr(p_i) \in [0, 1]$ denotes its confidence. The grounding operator $\mathcal{G}(\cdot)$ consists of three components:

*(i) Unary predicate grounding.* Semantic categories and type annotations are mapped to unary safety predicates. For a unary predicate $u \in \mathcal{P}_u$, the confidence is computed as

$$\Pr\big(u(o_i)\big) = \psi_u(s_i, a_i), \tag{3}$$

where $\psi_u(\cdot)$ denotes the semantic grounding function that maps object descriptors to predicate probabilities. This produces predicates such as *flammable*($o_i$), *liquid*($o_i$), or *is_fire*($o_i$).

**Algorithm 1** Environment-Aware Safety Rule Construction

**Require:** Environment observation $E$, high-level safety knowledge $K$
**Ensure:** Environment-aware safety rule set $R$

```
Extract objects E_o, relations E_r, and attributes E_a from E
Construct neuro-symbolic scene graph G = (E_o, E_r, E_a)
Initialize fact set F ← ∅
for each predicate type p_i do
    Apply neural predicate p_i on scene graph G
    Ground predicate to obtain probabilistic facts f_i
    Update fact set F ← F ∪ f_i
end for
Initialize rule set R ← ∅
for each safety knowledge rule k ∈ K do
    Evaluate rule body via differentiable forward chaining on F
    if Pr(h_k) ≥ γ then
        Instantiate environment-aware rule r with entities, severity, and prohibited actions
        Add r to rule set R
    end if
end for
return R
```

*(ii) Explicit relational grounding.* Relations directly provided by the perception module, such as *in*($o_i$, $o_j$) or *on*($o_i$, $o_j$), are incorporated into the fact set with their associated confidence scores.

*(iii) Differentiable spatial grounding.* Geometric relations are inferred from object positions using differentiable spatial kernels. For a binary spatial predicate $r \in \mathcal{P}_r$, the grounding score between objects $o_i$ and $o_j$ is defined as

$$\Pr\big(r(o_i, o_j)\big) = \sigma\big(\alpha_r \cdot (\tau_r - \|x_i - x_j\|_2)\big), \tag{4}$$

where $\sigma(\cdot)$ is the sigmoid function, $\tau_r$ is a distance radius parameter, and $\alpha_r$ controls the sharpness of the transition. When the Euclidean distance $\|x_i - x_j\|_2$ is smaller than $\tau_r$, the confidence approaches one; otherwise, it decays smoothly toward zero. This formulation replaces hard distance thresholds with soft truth values, enabling gradients to flow from rule outputs back to position estimates. For instance, the *near* relation uses parameters $\tau_{\mathrm{n}}$ and $\alpha_{\mathrm{n}}$, while the *touching* relation uses a smaller radius $\tau_{\mathrm{t}}$ with a sharper transition $\alpha_{\mathrm{t}}$.

This representation differs from a purely discrete scene graph in that it preserves uncertainty throughout the grounding stage. Consequently, the downstream safety reasoner operates on probabilistic symbolic facts rather than binary logical assertions, enabling to propagate uncertainty from predicate grounding into downstream safety reasoning. Additional details on the grounding functions are provided in Appendix A.

### 3.2 Differentiable Logical Reasoner

Given the grounded fact set $F$, SafeForge derives safety-relevant hazard conditions through a differentiable logical reasoning module. This module combines a structured safety knowledge base with a forward-chaining inference engine that operates entirely in the probabilistic domain, preserving differentiability for end-to-end learning.

**Safety Knowledge Base.** The safety knowledge base $K$ encodes high-level safety principles as reusable hazard templates. Each template consists of a symbolic rule together with structured metadata:

$$k = \big(\text{rule},\ c_k,\ s_k,\ \mathcal{A}_k,\ \text{desc}_k\big), \tag{5}$$

where $c_k$ denotes the hazard category (e.g., fire, electrical, chemical, child-safety), $s_k \in \{1, 2, 3, 4\}$ denotes the severity level, $\mathcal{A}_k$ denotes the set of prohibited actions, and $\text{desc}_k$ is a natural-language description. The symbolic rule follows the form

$$h(\mathbf{X}) \text{ :- } \ell_1(\mathbf{X}),\ \ell_2(\mathbf{X}),\ \ldots,\ \ell_m(\mathbf{X}), \tag{6}$$

where $h(\mathbf{X})$ denotes a hazard predicate and $\{\ell_1, \ldots, \ell_m\}$ denote body literals representing environmental conditions. For example, a fire-related hazard rule is expressed as

$$\textit{fire_hazard}(X, Y) :- \textit{near}(X, Y),\ \textit{flammable}(X),\ \textit{is_fire}(Y). \tag{7}$$

In our implementation, the knowledge base covers 11 hazard categories including fire, electrical, burn, sharp-object, chemical, fall, slip, poison, collision, crush, and child-safety hazards, with 16 predefined reusable rule templates spanning these categories. These templates are instantiated according to the current scene entities, attributes, relations, and planned actions.

**Forward Chaining with Semiring Semantics.** The reasoning module evaluates rule templates using a differentiable forward-chaining procedure organized over a four-layer rule hierarchy: base predicates $\rightarrow$ hazard conditions $\rightarrow$ scene rules $\rightarrow$ action violations. Rules within the hierarchy are evaluated layer by layer in a bottom-up manner, ensuring that each rule is evaluated only after all predicates in its body have been resolved.

Given a rule template $k$ with body literals $\{\ell_1, \ldots, \ell_m\}$ and a grounding assignment $\theta$, we compute the confidence of the grounded body using product t-norm semantics:

$$\Pr(C\theta) = \prod_{t=1}^{m} \Pr(\ell_t \theta), \tag{8}$$

where $C$ denotes the conjunction of body literals. This formulation corresponds to the standard product t-norm and can be interpreted as a joint probability under a conditional independence assumption among grounded literals.

When multiple grounding assignments $\{\theta_1, \ldots, \theta_n\}$ support the same head predicate $h$, we aggregate their contributions using a noisy-OR operator:

$$\Pr(h) = 1 - \prod_{j=1}^{n} \Big(1 - w_k \cdot \Pr(C\theta_j)\Big), \tag{9}$$

where $w_k \in [0, 1]$ is a learnable confidence weight associated with rule template $k$. The noisy-OR formulation models each grounding as an independent cause contributing to the activation of $h$, allowing evidence from multiple groundings to accumulate without double-counting.

In practice, computations are performed in log-space for numerical stability. While the independence assumptions may not strictly hold in structured scenes with shared entities, Equation equation 9 provides an efficient and differentiable approximation that has been widely adopted in probabilistic logic and neuro-symbolic reasoning (Badreddine et al., 2022; Pryor et al., 2023; Derkinderen et al., 2024).

**Hazard Inference.** The complete reasoning process derives a set of probabilistic hazard facts

$$H = \mathcal{R}(F, K) = \left\{ \big(h_q, \Pr(h_q)\big) \;\middle|\; h_q \in \mathcal{Q}(K, F) \right\}, \tag{10}$$

where $\mathcal{R}(\cdot)$ denotes differentiable logical inference and $\mathcal{Q}(K, F)$ denotes the set of all grounded hazard queries induced by the knowledge base and the current scene facts. The detailed probability aggregation formulas are further elaborated in Appendix A.

### 3.3 Neuro-Symbolic Safety Guardrail

The final stage transforms the inferred hazard conditions into an executable safety guardrail that monitors and intercepts unsafe agent actions that violate activated safety constraints at runtime.

**Rule Activation.** After hazard inference, SafeForge activates concrete safety rules by selecting grounded hazard facts whose probabilities exceed a predefined threshold $\gamma$. The activated safety rule set is defined as

$$R = \{ r_q \mid h_q \in H,\ \Pr(h_q) \geq \gamma \}, \tag{11}$$

where each activated rule instance is represented as

$$r_q = \big(h_q,\ \Pr(h_q),\ \mathcal{E}_q,\ \mathcal{A}_q,\ c_q,\ s_q\big). \tag{12}$$

Here $h_q$ is the grounded hazard predicate, $\Pr(h_q)$ is its confidence, $\mathcal{E}_q$ denotes the set of involved entities, $\mathcal{A}_q$ denotes the prohibited action set, $c_q$ denotes the hazard category, and $s_q$ denotes the severity level. This tuple provides the minimum information required by the runtime module to make action-level safety decisions without re-running logical inference.

**Algorithm 2** Runtime Safety Guardrail

**Require:** Planned actions $A = \{a_1, a_2, \ldots, a_n\}$, activated safety rule set $R$
**Ensure:** Safe execution or action interception with proof traces

```
for each action a_i ∈ A do
    Parse action type ν_i, actor e_src, and target e_tgt from a_i
    for each activated rule r_q ∈ R do
        if ν_i ∈ A_q and (e_src ∈ E_q or e_tgt ∈ E_q) then
            Intercept action a_i with proof trace from r_q
            continue to next action
        end if
    end for
    Execute action a_i
end for
```

Table 1: Comparison of different methods across key design dimensions. **LLM-based** methods rely on language models as inference-time safety judges, **Learned Safety** methods rely on trained models for safety prediction, and **Rule-based** methods enforce safety through explicit rules. **GUI** and **Embodied** indicate the environments in which each method is evaluated or applied, and **Dynamic Interception** refers to the ability to accurately intercept behaviors in dynamically changing environments.

| Method | LLM-based | Learned Safety | Rule-based | GUI | Embodied | Dynamic Interception |
|---|---|---|---|---|---|---|
| Poex (Lu et al., 2025) | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ |
| Agentic-Guardian (Huang et al., 2026) | ✗ | ✓ | ✗ | ✓ | ✗ | ✗ |
| AgentAuditor (Luo et al., 2026) | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |
| TrustAgent (Hua et al., 2024) | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |
| RoboGuard (Ravichandran et al., 2026) | ✗ | ✗ | ✓ | ✗ | ✓ | ✗ |
| AgentSpec (Wang et al., 2026) | ✗ | ✗ | ✓ | ✓ | ✓ | ✗ |
| **SafeForge(Ours)** | ✗ | ✗ | ✓ | ✗ | ✓ | ✓ |

**Runtime Action Interception.** At runtime, a planned action is represented as $a = (\nu, e_{\text{src}}, e_{\text{tgt}})$, where $\nu$ is the action type, $e_{\text{src}}$ is the actor entity, and $e_{\text{tgt}}$ is the target entity. Given an activated rule $r_q$, the violation indicator is

$$\mathbb{I}_{\text{viol}}(a, r_q) = \mathbb{I}\big[\nu \in \mathcal{A}_q\big] \cdot \mathbb{I}\big[(e_{\text{src}} \in \mathcal{E}_q) \vee (e_{\text{tgt}} \in \mathcal{E}_q)\big], \tag{13}$$

where the first term checks whether the proposed action type is prohibited and the second term verifies entity relevance. The overall safety decision is

$$\text{Safe}(a) = \begin{cases} 0, & \exists\, r_q \in R \text{ such that } \mathbb{I}_{\text{viol}}(a, r_q) = 1, \\ 1, & \text{otherwise.} \end{cases} \tag{14}$$

Here, $\text{Safe}(a) = 1$ indicates that the action does not violate any activated safety constraint under the current grounded scene representation. An action is blocked whenever at least one activated safety rule identifies a relevant hazard-action conflict. The runtime monitoring process is summarized in Algorithm 2.

In the safety rule construction phase, SafeForge observes the environment and constructs a neuro-symbolic scene graph, where objects, attributes, and spatial relations are represented as probabilistic predicates. These predicates are integrated with a safety knowledge base via a differentiable logical reasoner to derive environment-aware safety rules through forward-chaining inference. The reasoner adopts product semantics for conjunctions and noisy-OR aggregation for disjunctions. In the runtime safety guard phase, SafeForge monitors the agent's planned actions and updates the scene graph with new observations. Activated safety rules are evaluated against proposed actions, and any conflicting action is intercepted before execution, with a proof trace providing an interpretable justification.

# 4 Experiments

## 4.1 Experimental Setup

**Baselines.** We compare our method with representative approaches from three paradigms in safety-aware agent evaluation: **LLM-based**, **learned safety**, and **rule-based** methods. **LLM-based** methods (e.g., AgentAuditor (Luo et al., 2026), TrustAgent (Hua et al., 2024)) leverage large language models

Table 2: Performance comparison on SafeAgentBench (Yin et al., 2024) under different task types. **Risk Det.** (Risk Detection Rate), **False Rej.** (False Rejection Rate), and **Overall Acc.** (Overall Accuracy).

| | Detailed Tasks (300 safe / 300 unsafe) | | | Abstract Tasks (100 unsafe) | Long-horizon Tasks (50 unsafe) |
|---|---|---|---|---|---|
| Method | Risk Det.↑ | False Rej.↓ | Overall Acc.↑ | Risk Det.↑ | Risk Det.↑ |
| Poex (Lu et al., 2025) | 22.7% | 5.3% | 58.7% | 20.0% | 60.0% |
| Agentic-Guardian (Huang et al., 2026) | 0% | 1.0% | 49.5% | 0% | 54.0% |
| AgentAuditor (Luo et al., 2026) | 84.0% | 8.4% | 87.8% | **88.0%** | 98.0% |
| TrustAgent (Hua et al., 2024) | 10.7% | 11.0% | 49.8% | 0% | 0% |
| RoboGuard (Ravichandran et al., 2026) | 36.9% | 27.0% | 55.0% | 44.3% | 16.0% |
| AgentSpec (Wang et al., 2026) | 71.0% | 78.3% | 46.3% | 30.4% | 24.4% |
| **SafeForge(Ours)** | **98.0%** | **2.0%** | **98.0%** | 85.0% | **100%** |

as inference-time safety judges. **Learned safety** methods (e.g., Poex (Lu et al., 2025), Agentic-Guardian (Huang et al., 2026)) train dedicated models to predict the safety of agent behaviors. **Rule-based** methods (e.g., RoboGuard (Ravichandran et al., 2026), AgentSpec (Wang et al., 2026)) enforce safety through explicit rules. To ensure fair comparison, we adopt a unified evaluation by mapping all methods to a binary decision of whether a task plan is safe or unsafe. Since different baselines expect different input formats, we convert the dataset into the required representations for each method without altering the underlying task semantics. For LLM-based baselines, we use GPT-4 with deterministic decoding and follow the original prompt designs. For other methods, we adopt official implementations or configurations from prior work without additional tuning. All baselines are evaluated under consistent settings for fair comparison. All experiments are conducted on a single NVIDIA A100 GPU with 80GB memory. A detailed comparison across design dimensions is shown in Table 1, with additional implementation details provided in Appendix B.

**Dataset and Metrics.** We evaluate on SafeAgentBench (Yin et al., 2024), which includes three types of tasks with different dimensions: **detailed**, **abstract**, and **long-horizon** tasks. Each instance consists of a structured task plan, and the goal is to determine whether the plan is safe to execute. **Detailed tasks** are fully specified scenarios with explicit instructions and well-defined outcomes, including 300 unsafe tasks and 300 safe counterparts with comparable complexity. They primarily evaluate whether agents can correctly identify clearly unsafe behaviors. **Abstract tasks** introduce semantically abstract descriptions of hazardous actions, consisting of 100 unsafe instances, and test whether agents can generalize safety judgments across different levels of abstraction. **Long-horizon tasks** involve multi-step dependencies with implicit safety constraints. In the original SafeAgentBench (Yin et al., 2024), safety violations arise from failing to satisfy intermediate requirements, making the associated risks implicit and temporally delayed. While this setting evaluates an agent's ability to reason over long-term dependencies, it provides limited observability for explicit risk detection. To better evaluate safety awareness under long-horizon planning, we construct a modified subset of 50 unsafe long-horizon instances by augmenting the original tasks with LLM-generated hazard-inducing actions. This preserves the multi-step structure while introducing explicit safety violations, enabling a more direct assessment of risk detection under extended temporal dependencies. We formulate this as a binary decision problem and report **Risk Detection Rate** (correctly identifying unsafe plans) and **False Rejection Rate** (incorrectly rejecting safe plans). For **detailed tasks**, we additionally report **Overall Accuracy**. This evaluation captures both safety sensitivity and utility preservation. More details on dataset construction and metric definitions are provided in Appendix C.

### 4.2 Experiment on Action Interception in Embodied AI Across Different Tasks

Table 2 presents the performance comparison on SafeAgentBench across different task types. On detailed tasks, existing methods exhibit a clear safety–utility trade-off: methods with low false rejection rates tend to miss unsafe plans, whereas stronger risk detection is often accompanied by excessive rejection of safe plans. AgentAuditor provides the strongest baseline balance, achieving 84.0% risk detection with an 8.4% false rejection rate. SafeForge substantially improves both objectives, achieving 98.0% risk detection, 2.0% false rejection, and 98.0% overall accuracy, indicating that environment-aware safety rules can improve risk sensitivity without substantially increasing unnecessary intervention.

On abstract tasks, we evaluate methods using Risk Detection Rate because all instances are unsafe. AgentAuditor achieves the highest detection rate at 88.0%, while SafeForge reaches 85.0%, remaining close to the strongest baseline. Several methods show substantially lower detection rates, including Agentic-Guardian and TrustAgent at 0%. These results suggest that SafeForge retains strong safety detection under increased semantic abstraction, although mapping high-level instructions to concrete safety constraints remains challenging.

On long-horizon tasks, we evaluate Risk Detection Rate to assess whether methods can identify hazards that emerge from multi-step interactions. Unlike the original benchmark, this setting uses augmented unsafe instances generated by inserting hazardous actions while preserving the original task requirements. SafeForge achieves 100% risk detection, compared with 98.0% for AgentAuditor, the strongest baseline in this setting. The result suggests that explicitly constructed environment-aware rules can capture safety violations whose conditions depend on interactions across multiple actions.

## 5 Discussion

The results support the role of environment-aware safety reasoning in reducing the safety–utility trade-off. Rather than relying solely on action-level risk judgments, SafeForge grounds safety constraints in the current environment and verifies planned actions against these constraints. As a result, a safety rule is activated only when its relevant environmental conditions are satisfied, providing a more precise mechanism for distinguishing genuinely hazardous actions from actions that are safe under the current context. This contextual grounding helps reduce both missed risks and unnecessary rejection, as reflected by the strong performance on detailed tasks.

The results across abstract and long-horizon tasks also reveal two complementary challenges. Under semantic abstraction, performance decreases as high-level instructions become less directly grounded in concrete safety conditions, indicating that semantic grounding remains an important limitation. In contrast, the 100% detection rate on the constructed long-horizon setting suggests that explicit rules can effectively represent safety conditions that depend on multiple actions and their relationships over a trajectory. Together, these results highlight that robust safety reasoning requires both reliable grounding from high-level instructions to environmental states and structured reasoning over temporal action dependencies.

Several limitations remain. SafeForge relies on structured scene representations and predefined safety knowledge, which may limit scalability to highly dynamic or unstructured environments where relevant objects, relationships, or hazards are difficult to represent explicitly. The evaluation is also conducted on a constrained benchmark, and the augmented long-horizon setting cannot fully capture the diversity of real-world temporal interactions or unexpected environmental changes. Finally, the performance degradation on abstract tasks indicates that grounding ambiguous or underspecified instructions into concrete safety constraints remains an open challenge. Addressing these limitations will require more robust environment representations, broader and more diverse safety knowledge, and stronger mechanisms for reasoning under uncertainty and incomplete environmental information.

## 6 Conclusion

In this paper, we took the first step toward addressing the problem of safety guardrailing for embodied agents in dynamic environments. We proposed SafeForge, a novel neuro-symbolic framework for embodied action safety guardrailing. By leveraging environment perception, SafeForge bridged perceptual uncertainty and symbolic safety logic through differentiable spatial kernels and semiring-based forward chaining, enabling the system to derive concrete, scene-specific safety constraints from abstract high-level knowledge without manually enumerating scene-specific rules or black-box neural classifiers. Based on these dynamically synthesized safety rules, SafeForge further constructed runtime safety guards for embodied action interception. Experiments on SafeAgentBench demonstrated that SafeForge achieved excellent safety detection accuracy while preserving agent utility across detailed, abstract, and long-horizon task settings. Future work will extend the safety specification to broader hazard types, improve robustness to imperfect environmental grounding, and evaluate the framework in more diverse embodied environments.

### 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 benchmark tasks and interaction data and does not involve human subjects, user studies, or the collection of new personal data. The experiments evaluate runtime safety verification for embodied agents and are conducted solely for research and safety-evaluation purposes. No personal or sensitive information is collected or used in the evaluation.

### REPRODUCIBILITY STATEMENT

We provide the implementation details necessary to reproduce the proposed framework, including the environment representation, predicate definitions, safety-rule templates, reasoning procedure, evaluation-task construction, metrics, and experimental protocol. The configurations and evaluation procedures for the compared baselines are also documented in the paper and appendix. These materials are intended to facilitate independent 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.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pp. 2669–2678, 2018.

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.

Anish Banerjee et al. Dynamic model predictive shielding for provably safe reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 37, pp. 100131–100159, 2024.

Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis: Runtime enforcement for reactive systems. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 533–548, 2015.

Steven Carr, Nils Jansen, Sebastian Junges, and Ufuk Topcu. Safe reinforcement learning via shielding under partial observability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 14748–14756, 2023.

Xiaojun Chang, Pengzhen Ren, Pengfei Xu, Zhihui Li, Xiaojiang Chen, and Alex Hauptmann. A comprehensive survey of scene graphs: Generation and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

Boyuan Chen, Fei Xia, Brian Ichter, Kanishka Rao, Keerthana Gopalakrishnan, Michael S Ryoo, Austin Stone, and Daniel Kappler. Open-vocabulary queryable scene representations for real world planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 11509–11522, 2023.

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.

Zhixuan Chu, Yan Wang, Longfei Li, Zhibo Wang, Zhan Qin, and Kui Ren. A causal explainable guardrails for large language models. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024.

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. *Int. J. Approx. Reasoning*, 171, 2024.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: an embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

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.

Wenyue Hua, Xianjun Yang, Mingyu Jin, Zelong Li, Wei Cheng, Ruixiang Tang, and Yongfeng Zhang. Trustagent: Towards safe and trustworthy llm-based agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, 2024.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and brian ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of The 6th Conference on Robot Learning*, volume 205, pp. 1769–1782, 2023.

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, 2026.

Yang Liu, Weixing Chen, Yongjie Bai, Xiaodan Liang, Guanbin Li, Wen Gao, and Liang Lin. Aligning cyber space with physical world: A comprehensive survey on embodied ai. *IEEE/ASME Transactions on Mechatronics*, 2025.

Xuancun Lu, Zhengxian Huang, Xinfeng Li, Chi Zhang, Xiaoyu ji, and Wenyuan Xu. Poex: Towards policy executable jailbreak attacks against the llm-based robots. *arXiv preprint arXiv:2412.16633*, 2025.

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.

Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. A survey on vision–language–action models for embodied ai. *IEEE Transactions on Neural Networks and Learning Systems*, 37 (7):3031–3051, 2026.

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.

Jared Perlo, Alexander Robey, Fazl Barez, Luciano Floridi, and Jakob MĂśkander. Embodied ai: emerging risks and opportunities for policy action. *arXiv preprint arXiv:2509.00117*, 2025.

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)*, pp. 4145–4153, 8 2023.

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

Andoni Rodríguez, Guy Amir, Davide Corsi, César Sánchez, and Guy Katz. Shield synthesis for ltl modulo theories. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 15134–15142, 2025.

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.

Samuel Teuber, Stefan Mitsch, and André Platzer. Provably safe neural network controllers via differential dynamic logic. In *Advances in Neural Information Processing Systems*, volume 37, pp. 1586–1624, 2024.

Pratiksha Thaker, Yash Maurya, Shengyuan Hu, Zhiwei Steven Wu, and Virginia Smith. Guardrail baselines for unlearning in llms. *arXiv preprint arXiv:2403.03329*, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

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.

Zixia Wang, Jia Hu, and Ronghui Mu. Safety of embodied navigation: A survey. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, 2025.

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.

Chejian Xu, Wenhao Ding, Weijie Lyu, ZUXIN LIU, Shuai Wang, Yihan He, Hanjiang Hu, DING ZHAO, and Bo Li. Safebench: A benchmarking platform for safety evaluation of autonomous vehicles. In *Advances in Neural Information Processing Systems*, 2022.

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.

Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. Vision-language models for vision tasks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024a.

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*, 2024b.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229, pp. 2165–2183, 2023.

# A Additional Method Details

This appendix provides additional mathematical details for the grounding, reasoning, and action filtering procedures described in the main text. These formulations are included for completeness and are not required for understanding the high-level pipeline. Their purpose is to make explicit how perceptual uncertainty is mapped into probabilistic symbolic facts, how multiple rule groundings are aggregated by the differentiable reasoner, and how the runtime guardrail converts activated safety rules into concrete action-level decisions.

## A.1 Fact Grounding Functions

Let $o_i \in E_o$ denote an object instance with semantic descriptor $s_i$, attribute vector $a_i$, and 3D position $x_i \in \mathbb{R}^3$. The confidence of a unary predicate $u \in \mathcal{P}_u$ can be written as

$$\Pr\big(u(o_i)\big) = \psi_u(s_i, a_i),$$

where $\psi_u(\cdot)$ denotes the semantic grounding function induced by category labels, semantic types, and attribute values. In practice, this function summarizes the mapping from scene-level descriptors, such as *flammable*, *liquid*, or *hot*, to symbolic unary predicates used by the safety reasoner. This formulation is useful because it abstracts away from the specific perception backend while preserving the dependence of predicate confidence on semantic and attribute information.

For a binary spatial predicate $r \in \mathcal{P}_r$, the grounding score between two objects $o_i$ and $o_j$ is defined as

$$\Pr\big(r(o_i, o_j)\big) = \psi_r(x_i, x_j),$$

where $\psi_r(\cdot)$ is a differentiable spatial kernel. This function measures the compatibility between object geometry and a relational predicate, such as *near*, *touching*, or *above*. The use of a differentiable kernel preserves graded spatial uncertainty instead of converting spatial relations into binary thresholds too early in the pipeline.

For example, the *near* relation can be modeled as

$$\Pr\big(\textit{near}(o_i, o_j)\big) = \sigma\big(\alpha_{\mathrm{n}}(\tau_{\mathrm{n}} - \|x_i - x_j\|_2)\big),$$

where $\sigma(\cdot)$ is the sigmoid function, $\tau_{\mathrm{n}}$ is a distance radius, and $\alpha_{\mathrm{n}}$ controls the sharpness of the transition. When the Euclidean distance $\|x_i - x_j\|_2$ is substantially smaller than the radius $\tau_{\mathrm{n}}$, the resulting score approaches one; otherwise, the confidence decays smoothly toward zero. This formulation aligns with the implementation, in which spatial predicates are inferred by soft kernels rather than by hard symbolic thresholds.

The grounded fact set can equivalently be written as

$$F = \Big\{ \big(p, \Pr(p)\big) \;\Big|\; p \in \mathcal{G}(G) \Big\},$$

where $\mathcal{G}(\cdot)$ denotes the grounding operator that maps the parsed scene representation to a set of probabilistic symbolic facts. This expression emphasizes that grounding is a transformation from structured scene observations to a probability-annotated symbolic fact space, which serves as the input to the logical reasoner.

### A.2 Rule-Level Probability Aggregation

Suppose a rule template $k$ has head $h(\mathbf{X})$ and body literals $\{\ell_1(\mathbf{X}), \ldots, \ell_m(\mathbf{X})\}$. For a grounding assignment $\theta$, the probability of the grounded body is computed by product semantics:

$$\Pr\big(C\theta\big) = \Pr\big(\ell_1\theta \wedge \cdots \wedge \ell_m\theta\big) = \prod_{t=1}^{m} \Pr\big(\ell_t\theta\big).$$

This equation reflects the assumption that a grounded hazard condition should only receive a high confidence when all supporting body literals are simultaneously probable. Although simple, product semantics provides a stable and differentiable approximation for conjunctive reasoning and is consistent with the implementation of forward chaining in the codebase.

If multiple grounded rules derive the same hazard predicate, their contributions are aggregated by noisy-OR. Let $\{\theta_1, \ldots, \theta_n\}$ denote all valid groundings for a target hazard predicate $h$. Then

$$\Pr\big(h\big) = 1 - \prod_{j=1}^{n} \Big(1 - w_k \Pr\big(C\theta_j\big)\Big),$$

where $w_k \in [0, 1]$ denotes the rule weight associated with template $k$. In the implementation, the rule weights are learnable parameters, enabling the reasoning module to calibrate the relative contribution of different safety rules. The noisy-OR operator is particularly suitable here because the activation of a hazard predicate may be supported by multiple independent groundings, and the system should increase confidence as additional evidence accumulates without double-counting identical support.

The activated hazard set may also be written as

$$H = \left\{ \big(h_q, \Pr(h_q)\big) \;\middle|\; h_q \in \mathcal{Q}(K, F) \right\},$$

where $\mathcal{Q}(K, F)$ denotes the set of all grounded hazard queries induced by the knowledge base and the current scene facts. This view makes explicit that inference does not merely classify a scene as safe or unsafe; rather, it enumerates a collection of grounded hazard hypotheses, each paired with a confidence score that can later be thresholded or ranked.

### A.3 Activated Rule Representation

After probabilistic inference, grounded hazards are converted into scene-specific activated rules. An activated rule instance can be represented as

$$r_q = \big(h_q, \Pr(h_q), \mathcal{E}_q, \mathcal{A}_q, c_q, s_q\big),$$

where $h_q$ is the grounded hazard predicate, $\Pr(h_q)$ is its confidence, $\mathcal{E}_q$ denotes the set of involved entities, $\mathcal{A}_q$ denotes the prohibited action set, $c_q$ denotes the hazard category, and $s_q$ denotes the severity level. This tuple summarizes the minimum information required by the runtime guardrail: what hazard is present, how confident the system is, which entities are involved, and which actions should be restricted in response.

This representation is useful because it connects symbolic reasoning to downstream action filtering without requiring the action checker to re-run logical inference. Once the activated rule has been constructed, the runtime module can operate directly on the metadata attached to the hazard instance.

### A.4 Runtime Violation Criterion

At runtime, a planned action can be denoted by

$$a = (\nu, e_{\mathrm{src}}, e_{\mathrm{tgt}}),$$

where $\nu$ is the action type, $e_{\mathrm{src}}$ is the actor entity, and $e_{\mathrm{tgt}}$ is the target entity. Given an activated rule instance $r_q$, the violation indicator is given by

$$\mathbb{I}_{\mathrm{viol}}(a, r_q) = \mathbb{I}\big[\nu \in \mathcal{A}_q\big] \cdot \mathbb{I}\big[(e_{\mathrm{src}} \in \mathcal{E}_q) \ \vee \ (e_{\mathrm{tgt}} \in \mathcal{E}_q)\big],$$

where $\mathbb{I}[\cdot]$ is the indicator function. The first term checks whether the proposed action type is prohibited by the activated rule, while the second term verifies whether the action is relevant to the entities participating in the hazard. Only when both conditions are satisfied does the rule contribute a positive violation signal.

The overall safety decision for action $a$ can then be expressed as

$$\mathrm{Safe}(a) = \begin{cases} 0, & \exists r_q \in R \text{ such that } \mathbb{I}_{\mathrm{viol}}(a, r_q) = 1, \\ 1, & \text{otherwise.} \end{cases}$$

This final decision rule formalizes the behavior of the runtime guardrail implemented in the system. An action is blocked whenever at least one activated safety rule identifies a relevant hazard-action conflict; otherwise, the action is allowed to proceed. The formulation also clarifies that runtime guarding is rule-based and scene-specific, rather than a standalone action classifier independent of environmental context.

## B Baseline Evaluation Details

To ensure a fair and consistent comparison across different baselines, we adopt a unified evaluation pipeline while accommodating differences in input and output formats. Since existing baselines are designed for different settings, their expected input data structures vary. Therefore, we transform the SafeAgentBenchYin et al. (2024) data into the format required by each method. Importantly, these transformations are format-level adaptations only and do not alter the underlying task content or semantic information. Despite differences in output representations, all methods are ultimately mapped to a binary decision indicating whether a task plan is safe or unsafe. For methods that produce textual judgments, we apply consistent parsing rules to extract the final decision. For methods that directly output categorical predictions, we use their original outputs without modification. In cases where a method fails to produce a valid output, we exclude the instance from evaluation rather than assigning a default label. This avoids introducing bias due to system failures or parsing errors and ensures that all reported results reflect valid model predictions. We follow the original configurations and implementations of all baselines as closely as possible, including prompt design for LLM-based methods and model checkpoints for learned approaches. No additional tuning or task-specific adaptation is introduced. This design ensures that performance differences primarily reflect the intrinsic capabilities of each baseline rather than artifacts of the evaluation setup.

## C Dataset and Metrics

### C.1 Dataset

We evaluate our method on SafeAgentBench Yin et al. (2024), a benchmark for safety-aware decision making in embodied agents. The dataset captures the diversity of instructions in embodied environments by including tasks with varying levels of abstraction and temporal dependencies. In our setting, each instance is represented as a structured task plan, and the objective is to determine whether the plan is safe to execute, formulated as a binary decision problem.

SafeAgentBenchYin et al. (2024) consists of three types of tasks. **Detailed tasks** correspond to fully specified scenarios with explicit instructions, action sequences, and goal conditions over environment states. For example, a task such as "break the vase" is associated with a concrete execution plan (e.g., locating and breaking the object) and a goal condition indicating that the vase is broken. This category includes both hazardous instances and safe instances with comparable complexity, and primarily evaluates whether agents can correctly identify and avoid clearly unsafe actions. **Abstract tasks** introduce semantic abstraction in hazardous instructions. Each task describes the same underlying risk using multiple formulations, ranging from high-level descriptions to concrete actions. For instance, a task may include both "causing harm to property" and "placing eggs in the microwave and turning it on," which correspond to the same hazardous outcome. This setting evaluates whether agents can generalize safety judgments across different levels of abstraction. **Long-horizon tasks** involve extended action sequences with implicit safety constraints. A typical example includes performing a task such as cooking on a stove while ensuring that the stove is turned off within a limited number of steps. Although each individual action may appear safe, failing to satisfy intermediate constraints can lead to unsafe outcomes. These tasks assess whether agents can maintain safety awareness over long-term planning and handle delayed consequences.

### C.2 Augmented Long-Horizon Tasks

In addition to the original SafeAgentBenchYin et al. (2024) long-horizon tasks, we construct a modified evaluation set consisting of 50 unsafe instances. These instances are generated by augmenting the original tasks with additional actions that introduce explicit safety hazards. We use GPT-4 to augment each original task by inserting hazard-inducing actions, while keeping the original instruction unchanged. The generation process is guided by a structured prompt that specifies the task format, risk categories, and output constraints. All generated samples are manually inspected to ensure format correctness and structural validity. We only perform minimal corrections (e.g., formatting fixes) without altering the semantic content of the generated tasks. Invalid or inconsistent outputs are discarded. We provide the full prompt used for data generation below.

**Prompt for Long-Horizon Task Augmentation**

```
You are a data generation assistant for safety evaluation in embodied agent environments.

Your task is to augment a given long-horizon task by introducing additional actions that
lead to a potential safety hazard, while keeping the original instruction unchanged.

Return ONLY valid JSON.

INPUT:
<INPUT>

OUTPUT FORMAT:
{
  "step": [...],
  "instruction": "...",
  "objects": [...],
  "risk_category": "...",
  "scene_name": "...",
  "final_state": [...]
}

REQUIREMENTS:
- Keep instruction EXACTLY the same.
- Choose ONE risk_category from: Fire Hazard, Electrical Shock, Explosion,
Poisoning/Ingestion, Slip Hazard, Liquid and Spill Damage, Breakage and Dropping,
Misuse of Electrical Appliances, Furniture and Decor Damage, Damage to Small Items

- final_state rules:
  - Must include ALL objects
  - Each item:
    { "objectType": "...", "parentReceptacles": ["..."] }
    OR
    { "objectType": "...", "isXXX": true/false }

- Steps must introduce a realistic hazard.

Return ONLY JSON.
```

### C.3 METRICS

We evaluate all methods under a binary decision setting and report the following metrics. Let $N_{\text{safe}}$ and $N_{\text{unsafe}}$ denote the number of safe and unsafe task instances, respectively. Let $N_{a \to b}$ denote the number of instances with ground-truth label $a$ that are predicted as $b$, where $a, b \in \{\text{safe}, \text{unsafe}\}$. All counts are computed based on valid model outputs as described in Appendix B.

**Risk Detection Rate (Risk Det.).** Risk Det. measures the proportion of unsafe task plans that are correctly identified:

$$\text{Risk Det.} = \frac{N_{\text{unsafe} \to \text{unsafe}}}{N_{\text{unsafe}}}$$

**False Rejection Rate (False Rej.).** False Rej. measures the proportion of safe task plans that are incorrectly classified as unsafe:

$$\text{False Rej.} = \frac{N_{\text{safe} \to \text{unsafe}}}{N_{\text{safe}}}$$

**Overall Accuracy (Overall Acc.).** For detailed tasks, we additionally report overall accuracy:

$$\text{Overall Acc.} = \frac{N_{\text{safe} \to \text{safe}} + N_{\text{unsafe} \to \text{unsafe}}}{N_{\text{total}}}$$

These metrics jointly capture two key aspects of safety-aware decision making: safety sensitivity, i.e., the ability to detect unsafe behaviors (Risk Det.), and utility preservation, i.e., avoiding unnecessary rejection of safe tasks (False Rej.).