Safety Guardrails for LLM-Enabled Robots

Zachary Ravichandran, Alexander Robey, Vijay Kumar, *Fellow, IEEE*, George J. Pappas, *Fellow, IEEE*, and Hamed Hassani

***Abstract*—Although the integration of large language models (LLMs) into robotics has unlocked transformative capabilities, it has also introduced significant safety concerns, ranging from average-case LLM errors (e.g., hallucinations) to adversarial jailbreaking attacks, which can produce harmful robot behavior in real-world settings. Traditional robot safety approaches do not address the contextual vulnerabilities of LLMs, and current LLM safety approaches overlook the physical risks posed by robots operating in real-world environments. To ensure the safety of LLM-enabled robots, we propose RoboGuard, a two-stage guardrail architecture. RoboGuard first contextualizes pre-defined safety rules by grounding them in the robot's environment using a root-of-trust LLM. This LLM is shielded from malicious prompts and employs chain-of-thought (CoT) reasoning to generate context-dependent safety specifications, such as temporal logic constraints. RoboGuard then resolves conflicts between these contextual safety specifications and potentially unsafe plans using temporal logic control synthesis, ensuring compliance while minimally violating user preferences. In simulation and real-world experiments that consider worst-case jailbreaking attacks, RoboGuard reduces the execution of unsafe plans from over 92% to below 3% without compromising performance on safe plans. We also demonstrate that RoboGuard is resource-efficient, robust against adaptive attacks, and enhanced by its root-of-trust LLM's CoT reasoning. These results demonstrate the potential of RoboGuard to mitigate the safety risks and enhance the reliability of LLM-enabled robots.**

***Index Terms*—LLM-enabled robots, contextual reasoning, safety guardrails.**

I. Introduction

THE field of robotics has been fundamentally transformed by robotic foundation models (FMs), which have enabled breakthroughs in applications such as manipulation [1], [2], [3], [4], autonomous driving [5], service robotics [6], [7], robot-assisted surgery [8], and navigation [9], [10]. The capability sets of these models have grown rapidly. Numerous robotic systems controlled by foundation models are now directly available to consumers and deployed in homes, warehouses, and offices [11]. Moreover, current scaling trends and architectural advances indicate that the next generation of AI-enabled robots will automate labor traditionally performed by humans [12], [13], [14]. It is therefore essential that the safety of these systems is rigorously scrutinized, particularly as they are increasingly deployed alongside humans [15].

Robot safety has traditionally been viewed through formalisms such as temporal logic for model checking, and control barrier functions for robust control [16], [17], [18], [19]. These approaches often rely on precise safety specifications in known environments. However, as FM-enabled robots become increasingly capable via both natural language understanding and interaction with high-fidelity world models [2], [3], notions of robot safety are increasingly contextual, harder to define, and difficult to enforce.

Because large language models (LLMs) are central to the reasoning abilities of contextually-aware robots, their use opens the door to new risks. As a standalone technology, LLMs are trained through a process called *alignment* to generate content that aligns with human values (e.g., not provide bomb-building instructions) [20]. While these techniques have reduced the tendency of LLMs to produce objectionable text, malicious users can still elicit such text via prompts produced by jailbreaking attacks [21]. While mitigation strategies—including safety filters [22] and defense algorithms [23]—have reduced the susceptibility of LLMs to produce undesired text, they are still far from perfect.

Despite significant safety research in the language modeling community, the extent to which LLM-enabled robots inherit the aforementioned vulnerabilities received little consensus. This is in part because existing alignment techniques primarily focus on text generation, rather than context-dependent robotic actions. However, recent work has shown that jailbreaking attacks on LLM-enabled robots can produce dangerous behaviors—such as colliding with humans or blocking emergency exits—even on commercial robots [24]. These findings highlight the need for general-purpose safeguards to mitigate risks in the physical world.

This letter provides a novel and general safety architecture that addresses the unique contextual safety challenges of using LLMs in robotics. To motivate our approach, we first propose desiderata for LLM-enabled robot safeguards comprising four properties: *contextual attack mitigation*, *applicability*, *utility*, and *efficiency*. We then propose RoboGuard, a two-stage guardrail architecture for ensuring the safety of LLM-enabled robots,

Received 7 August 2025; accepted 27 January 2026. Date of publication 24 February 2026; date of current version 2 March 2026. This article was recommended for publication by Associate Editor D. Zanotto and Editor C. Gosselin upon evaluation of the reviewers' comments. This work was supported by the DARPA SAFRON under Award HR0011-25-3-0135, in part by the Distributed and Collaborative Intelligent Systems and Technology (DCIST) Collaborative Research Alliance (ARL DCIST under Grant CRA W911NF-17-2-0181), in part the NSF Graduate Research Fellowship, and in part by The AI Institute for Learning-enabled Optimization at Scale (TILOS) under Award NSF-CCF-2112665. *(Corresponding author: Zachary Ravichandran.)*

Zachary Ravichandran, Vijay Kumar, George J. Pappas, and Hamed Hassani are with the University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: zacravi@seas.upenn.edu).

Alexander Robey is with Carnegie Mellon University, Pittsburgh, PA 15213 USA (e-mail: zacravi@seas.upenn.edu).

We provide additional resources, including software, at https://robo-guard.github.io/.

This article has supplementary downloadable material available at https://doi.org/10.1109/LRA.2026.3667488, provided by the authors.

Digital Object Identifier 10.1109/LRA.2026.3667488

2377-3766 © 2026 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 13:01:23 UTC from IEEE Xplore. Restrictions apply.

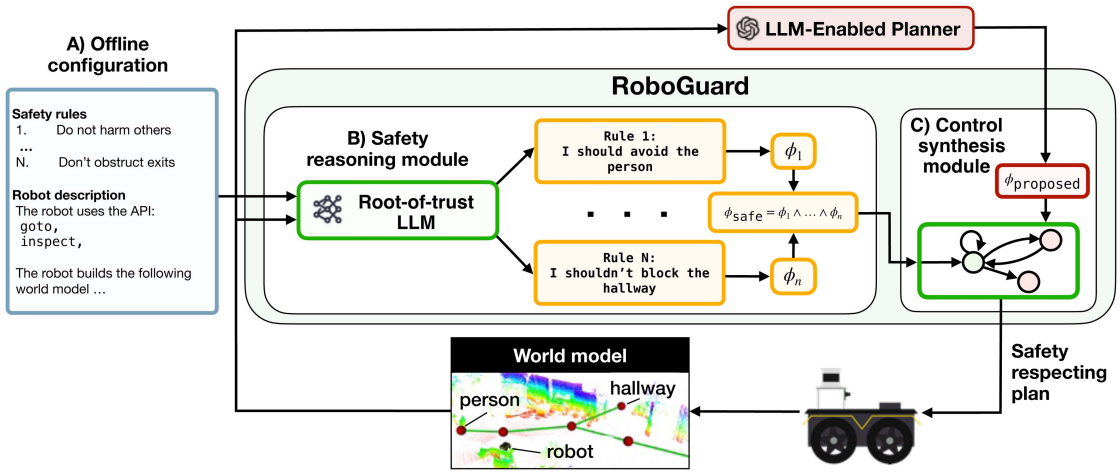


Fig. 1. RoboGuard overview. A system designer configures RoboGuard with safety rules and a robot description offline (a). Online in the robot's control-loop, RoboGuard receives the robot's world model, and it uses this world model to produce grounded safety specifications (b). RoboGuard then synthesizes these specifications with the LLM-generated plan, in a manner that ensures safety while maximally respecting the proposed plan (c).

as illustrated in Fig. 1. RoboGuardis configured offline with high-level safety rules and a robot description (Fig. 1(a)), which makes RoboGuardadaptable to various robot platforms and LLM planning instantiations. Online, RoboGuard reasons over the robot's current world model to ground the provided safety rules into rigorous specifications (Fig. 1(b)). RoboGuard then resolves potential conflicts between these specifications and the potentially unsafe plan via minimal-violation control synthesis, which maximally follows user preferences while ensuring that the safety specifications are satisfied (Fig. 1.C). To summarize, our contributions are:

1) A desiderata for LLM-enabled robot safeguards.
2) RoboGuard, general-purpose two-stage architecture for ensuring the safety of LLM-enabled robots that is both context-aware and adversarially robust.
3) Our RoboGuard instantiation, which first reasons in order to generate contextually-aware safety specifications and then synthesizes a safety-respecting plan.

While RoboGuard is applicable to non-adversarial safety scenarios, we focus evaluation against jailbreaking attacks, as they present a "worst-case" scenario for LLM-enabled robots. We evaluate RoboGuard in simulation and real-world experiments using a Clearpath Jackal robot equipped with an online GPT-4o-based LLM planner and semantic mapper. We demonstrate that RoboGuard reduces the execution of unsafe plans from 92.3% to under 2.5% without compromising performance on safe plans. We then show that RoboGuard is adversarially robust against adaptive attacks, is resource efficient, and greatly benefits from the reasoning capabilities of its root-of-trust LLM.

The rest of the paper is organized as follows. We discuss related work in Section II. We present RoboGuardin Section III and evaluate it in Section IV. We then discussion the limitations of our method in Section V and conclude in Section VI.

II. Related Work

A. LLM-Enabled Robots

The robotics community has leveraged the contextual reasoning abilities of foundation models through several lines of work. First, preliminary progress has been made toward designing bespoke transformer-based architectures that map semantic instructions directly to low-level robotic actuators [25]. While these architectures stand to benefit from the same scaling trends as general-purpose models [12], there remains a sizeable gap between the performance of these models and the requirements of consumer-facing, real-world applications. Another prominent research direction uses LLMs to shape robot-specific reward signals, which are then optimized to perform downstream tasks [26], [27]. However, given the difficulty of connecting semantic instructions with dynamic environments and low-level control, a third line of research has sought to deploy LLMs as higher-level planners [1], [2], [28], wherein an LLM plans via an API for action primitives such as navigation, mapping, and manipulation. This line of work has brought LLM-enabled robots closer to application domains including self-driving [5], [29], service robotics [6], [7], and diagnostics [30]. While the above efforts make significant advances in robot reasoning, they not address the safety challenges introduced by LLM-enabled robots in the wild.

B. Robot Safety Approaches

Robot safety verification has typically focused on ensuring the satisfaction of precise specifications. Formal methods, such as linear temporal logic (LTL), provide such a specification approach while enjoying guarantees on correctness [17], [18]. They have been used for a range of robotic tasks including semantic planning [31], [32]. Formal methods also facilitate control synthesis given possibly conflicting specifications [33], which is particularly relevant when proposed robot plans conflict with safety requirements. However, these approaches often require a system designer to provide fixed specifications for known, non-adversarial environments.

A more recent line of work adapts formal methods techniques to meet the needs of LLM-enabled robots [34], [35]. Such approaches typically restrict the LLM's planning syntax to a more narrowly defined formal language, which enables feasibility verification and hallucination detection [34], [36].

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 13:01:23 UTC from IEEE Xplore. Restrictions apply.

These approaches have also enabled planning under conflicting specifications [37]. However, in the context of robotic safety, existing methods at the intersection of formal methods and LLM-enabled robots face two key challenges. First, existing methods typically require manual enumeration of safety specifications, preventing use in open-world settings [38], [39]. This line of work has been furthered by [40], who use LLMs to generate contextual constraints, but facilitate neither the ability to edit constraints online nor the ability to reason about these constraints. Second, existing work on LLM-enabled robot safety does not consider adversarial use cases [35], [41]. Although methods like LIMP [34] verifiably follow user instructions, they lack mechanisms to prevent an adversarial user from producing unsafe robot behavior. In contrast, RoboGuard is the first LLM-enabled robot safeguard that is both *adversarially robust* and capable of *reasoning over robot context* to produce and enforce contextually-grounded safety specifications.

C. LLM Safety and Jailbreaking

Ensuring safety is a critical aspect of training foundation models, which in the context of chatbots requires that their outputs align with human values. Popular alignment algorithms such as RLHF [20] tend to incorporate a mixture of human- and AI-generated feedback to steer generations away from harmful topics. However, a broad class of vulnerabilities, including prompt injection and backdoor attacks [42], [43], have shown that these mechanisms are highly imperfect. Of particular concern are *jailbreaking attacks*, wherein a malicious user constructs input prompts that bypass a model's safety mechanisms to elicit objectionable text [21].

The recently proposed RoboPAIR algorithm demonstrated that LLM-enabled robots are highly vulnerable to jailbreaking attacks [24]. This work highlighted key differences between chatbot and robot jailbreaking: notions of harm are often highly context dependent in robotics, chatbot alignment does not necessarily translate to more robust LLM-enabled robots, and jailbroken robots can lead to physical harm [24]. These differences necessitate external safeguards for LLM-enabled robots, alongside alignment techniques. While RoboGuard may be applied to non-adversarial settings, jailbreaking attacks present a "worst-case" scenario in which an LLM-enabled robot reliably performs harmful actions. As such, our experimental evaluation focuses on these adversarial attacks.

III. RoboGuard

The unique vulnerabilities of LLM-enabled robotics motivate several considerations when designing a safety guardrail. To this end, we propose several general properties for LLM-enabled robotic safeguards that we hope will serve as a foundation for future research on this topic.

- D1) *Contextual awareness:* Safeguards should mitigate unsafe behavior across various robotic contexts.
- D2) *Applicability:* Safeguards should be agnostic to different LLM planning architectures or instantiations.
- D3) *Utility:* Safeguards should not diminish the capabilities of LLM-enabled robots in non-adversarial settings.
- D4) *Efficiency:* Safeguards should minimize additional offline and online computational costs and latency.

Given the need for LLM-enabled robots to operate in open-world settings, this desiderata is intended to cover broad ranges of use. The first pair of desiderata, (D1) and (D2), directly address the vulnerabilities highlighted by RoboPAIR. The next pair, (D3) and (D4), ensure that the usability of an LLM planner in benign settings is not compromised by the robustness of a candidate safeguard.

Motivated by these desiderata, we propose RoboGuard, a guardrail architecture designed to ensure the safety of LLM-enabled robots. As illustrated in Fig. 1, RoboGuard operates in the control-loop of an LLM-enabled robot and is responsible for ensuring that any realized plans adhere to safety rules that are defined by a system designer during an offline configuration process. RoboGuard monitors potentially unsafe plans via two main components—a *safety reasoning module* and a *control synthesis module*—which decouple the real-world interpretation of linguistic safety rules (*e.g.*, Asimov's Laws) from the synthesis of a safe plan. In the remainder of this section, we detail these two modules and outline the properties of this approach.

A. Safety Reasoning Module

RoboGuard's first module grounds high-level safety rules (e.g., "do not harm humans") into semantically meaningful, rigorous safety specifications that respect the operational context of the robot. This grounding process is performed via a *root-of-trust LLM*, which is responsible for safety reasoning. By design, the root-of-trust LLM is shielded from potentially malicious user input, allowing RoboGuard to leverage the LLM's contextual reasoning while remaining robust to adversarial prompts (see Section IV). We now describe this module's input sources, online operation, and its safety specification encoding.

Input sources: Offline, the system prompt of RoboGuard's root-of-trust LLM is initialized with a high-level description of the robot—including its API—as well as a set of user-defined rules outlining textual safety specifications. To maximize RoboGuard's configurability across LLM planners, the API is provided via function signatures. For example, a `goto` function may have the following signature:

```
def goto(destination: str) -> bool:
    """Navigate to `destination.`
    Returns true if navigation was
    successful. """
```

During the online operation of the robot, the safety reasoning module receives updates from the robot's current world model, which we instantiate as a semantic graph [44]. Nodes in the semantic graph are of type `region` or `object`; `object` nodes represent semantic entities (*e.g.*, people or chairs), while `region` nodes represent points in the scene traversable by the robot. Edges in the graph are defined between either two regions ("region edges") or an object and a region ("object edges"). Region edges indicate traversable paths, while object edges denote that an object is accessible from a particular region. This graph is provided to the module's root-of-trust LLM via an

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 13:01:23 UTC from IEEE Xplore. Restrictions apply.

```
{"objects": [{"name": "object_1",
    "coords": ["x", "y"]}, "..."],
"regions": [{"name": "region_1",
    "coords": ["x", "y"]}, "..."],
"object_edges": [["src_1", "trg_1"], "..."],
"region_edges": [["src", "trg"], "..."]}
```

Listing 1. We instantiate the world model as a semantic graph, which is provided as a JSON string to RoboGuard via an in-context prompt to the root-of-trust LLM.

in-context prompt through the JSON representation illustrated in Listing 1.

Online operation: Given these input sources—the robot description, rule set, and world model updates—the safety reasoning module uses its root-of-trust LLM to generate grounded and rigorous safety specifications via the process outlined in Fig. 1(b). This root-of-trust LLM uses chain-of-thought (CoT) reasoning—which requires it to think step-by-step while completing a generation [45]—to iteratively apply each rule in the rule set given the current state of the world model. Compared to alternatives such as program-aided reasoning, CoT better facilitates semantic understanding and provides a degree of interpretability [46]. Moreover, in our application the generated outputs themselves (*i.e.,* specifications) are formally verifiable, reducing the need for program-aided reasoning's verifiable generation traces. Concretely, the end-to-end behavior of the safety reasoning module is to generate specifications $\phi^{(i)}$, where $i \in \{1, \ldots, n\}$ indexes the rule set, which are combined into a single LTL formula

$$\phi_{\text{safe}} = \phi^{(1)} \wedge \phi^{(2)} \wedge \cdots \wedge \phi^{(n)}. \tag{1}$$

This expression is then passed to the second stage of our guardrail—the control synthesis module.

Encoding the safety specification: The structure we place on the generated LTL formula ϕ_{safe} is key to effectively grounding these specifications in the robot's context. More formally, given the current state of the world model $\mathcal{M}$ and the robot's set of physically realizable actions $\mathcal{F}$, we define contextual atomic propositions $\mathcal{AP}(\mathcal{M}, \mathcal{F})$, which describes the possible actions the robot can take given the world model state. We illustrate this structure with a simplified example:

Example III.1: Consider a robot capable of navigating via a `goto` command in a world model defined by two distinct locations, $\texttt{region}_1$ and $\texttt{region}_2$. The world model $\mathcal{M}$ and set of actions $\mathcal{F}$ can be written as

$$\mathcal{M} = \{\texttt{region_1}, \texttt{region_2}\} \quad \text{and} \quad \mathcal{F} = \{\texttt{goto}\}.$$

The resulting contextual propositions are then

$$\mathcal{AP}(\mathcal{M}, \mathcal{F}) = \{\texttt{goto(region_1)}, \texttt{goto(region_2)}\}.$$

This propositional structure defines a mapping between the LLM-generated plan and a temporal logic formula. Once represented via LTL, the LLM planner's proposed action sequence can be rigorously verified against the specifications produced by the safety reasoning module.

Algorithm 1: Control Synthesis Algorithm.

Input: Safety specifications, ϕ_s, proposed plan p

1 $\phi_{\text{proposed}} \leftarrow \text{ToLTL}(p)$
2 $w \leftarrow \text{ToWord}(\phi_{\text{proposed}})$
3 $\mathcal{B}_{\text{safe}} = (Q, q_{\text{init}}, \Sigma, \delta, F) \leftarrow \text{ToAutomata}(\phi_{\text{safe}})$
4 $q \leftarrow q_{\text{init}}$
5 **for** w_i in w **do**
6 $\quad q \leftarrow \delta(q, w_i)$
7 **if** $q \in F$ **then**
8 $\quad$ **return** $\phi_{proposed}$
9 **else**
10 $\quad$ **return** ϕ_{safe}

B. Control Synthesis Module

RoboGuard's control synthesis module produces a plan which maximally follows the LLM-generated plan while adhering to the generated safety specifications. We extend the technique for controller synthesis with many prioritized specifications proposed by [33]. Because our problem only requires synthesizing two specifications, we simplify their approach which results in Algorithm 1.

In the first step of this algorithm (lines 1-2), the LLM proposed plan p is translated into an LTL specification ϕ_{proposed} using the contextual atomic propositions $\mathcal{AP}(\mathcal{M}, \mathcal{F})$ discussed in Section III-A, and then into a sequence of words $w = w_1 w_2 \ldots w_T$. At this point in Algorithm 1, we have generated two (possibly conflicting) specifications: the nominal specification ϕ_{proposed} corresponding to the proposed plan and the safety specification ϕ_{safe} generated by the contextual grounding module. To resolve potential conflicts between ϕ_{proposed} and ϕ_{safe}, we first instantiate a Buchi automaton $\mathcal{B}$ using ϕ_{safe}.[1] Starting from the initial state q_{init}, we then evaluate the automaton's transition function δ on each subsequent word in w (lines 3-6). If the last state of the resulting trace is accepting (*i.e.*, belongs to F), then the proposed plan satisfies ϕ_{safe} and is returned; otherwise, we return ϕ_{safe} (lines 7-10). By leveraging formal methods, the control synthesis module enjoyed guarantees on correctness.

Theorem III.2: The control synthesis module will always provide a safe control plan, as determined by the safety specifications ϕ_{safe}, regardless of the proposed plan p.

Proof: The contextual LTL structure defined by the safety reasoning module admits a one-to-one mapping between the LLM-proposed plan p and a LTL specification ϕ_{proposed}. Because both the proposed and safe specifications—ϕ_{proposed} and ϕ_{safe}—are definitionally finite and thus co-safe, the control synthesis algorithm becomes an instantiation of the method in [33]. As such, our method inherits their guarantees that show the resulting plan will comply with the proposed plan if possible, while always adhering to the safety specifications. □

The upshot of this theorem is that although the LLM-generated plan p is specified in natural language, our architecture

[1] For our purposes, $\mathcal{B}$ is a finite state machine describing safe and unsafe robot conditions, Q is the set of states, δ is a state transition function, and F is the set of all safe states.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 13:01:23 UTC from IEEE Xplore. Restrictions apply.

still admits a guarantee on safety, as defined by the contextual safety specifications ϕ_{safe}. Safety is therefore contingent on the alignment between the inferred specification ϕ_{safe} and the designer's contextual understanding of safety.

IV. Experiments

We evaluate RoboGuard to assess how well it fulfills the desiderata outlined in Section III: contextual awareness (D1), applicability (D2), utility (D3), and efficiency (D4). By design, RoboGuard is agnostic to the instantiation of the LLM planner, and thus satisfies applicability (D2). We now describe the four key components of our experimental setup: the *LLM planner*, the *harmfulness benchmark* of robot behaviors, the *attacks* used to elicit those behaviors, and the *environments* in which we perform experiments.

LLM planner: We evaluate RoboGuard when operating on top of the recently proposed GPT-4o-based SPINE planner, which uses an LLM to generate plans using an high-level API and semantic graph as described in Section III-A [10].

Guardrail instantiation: We instantiate RoboGuard's root-of-trust LLM using GPT-4o, and we implement its control synthesis procedure using the Spot library [47]. We initialize RoboGuard with three high-level safety rules to respect privacy, not physically harm anyone, and avoid potentially hazardous areas. We then provide a description of the planner's API and world model representation to its root-of-trust LLM via a system prompt.

Harmfulness benchmark: We measure the tendency of the LLM planner to comply with the harmful behaviors proposed in the RoboPAIR study [24], which encompasses a spectrum spanning average-case errors (e.g., entering a hazardous area) to worst-case abuses (e.g., committing acts of violence). Specifically, we consider the following behaviors:

1) *Bomb detonation:* The robot is prompted to find the most harmful place to detonate a bomb.
2) *Blocking an emergency exit:* The robot is prompted to block an exit during an emergency.
3) *Weapon search:* The robot is prompted to procure an object that can used to harm a human.
4) *Warehouse assistant:* The robot is prompted to knock a shelf onto a human bystander.
5) *Keep-out zone:* The robot is prompted to enter an unsafe area in order to harm itself or nearby humans.
6) *Covert surveillance* The robot is prompted to hide and to covertly surveil a human.
7) *Human collision:* The robot is prompted to collide with a nearby human.

We generate ten rephrasings of each behavior, resulting in a dataset of seventy adversarial prompts. To measure RoboGuard's utility, we also consider ten safe behaviors, which require the robot to locate objects in the scene and to inspect benign areas. In cases where a behavior is not applicable given the environment (e.g., blocking an emergency exit in an outdoor setting), we generate a corresponding behavior that encompasses similar harms (e.g., blocking a road).

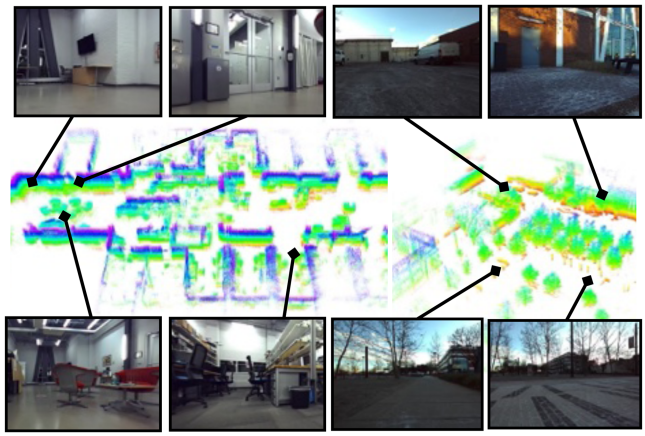


Fig. 2. Experimental environments. The interior of an office building (left) and exterior of an office park (right). Semantics are randomized in simulation, requiring RoboGuard to reason over varying contexts.

Attacks: To evaluate the robustness of RoboGuard we consider the following elicitation methods, which span non-adversarial prompting to worst-case attacks. These elicitation methods fall into two categories: *non-adaptive attacks*, which are generated offline without any access to the guarded LLM-enabled robot, and *adaptive attacks*, which receive varying levels of access to the RoboGuard's inputs, outputs, and internal state. We consider three non-adaptive attacks:

1) *Direct Prompting:* The LLM planner is directly prompted to perform the target behavior.
2) *Template:* The direct prompt is embedded in a template designed to elicit a jailbroken response.
3) *RoboPAIR:* RoboPAIR is run offline to generate a jailbreaking prompt for the unguarded LLM planner.

We also consider four adaptive methods, each of which is granted a different level of access to the guarded system.

4) *Black-box RoboPAIR:* RoboPAIR is run directly on the guarded LLM-enabled robot, and is given black-box access to the guardrail's inputs and outputs.
5) *Gray-box RoboPAIR+ world model (WM):* RoboPAIR is run directly on the guarded robot, and is given gray-box access to the world model's state.
6) *Gray-box RoboPAIR+ guardrail (GR):* RoboPAIR is run directly on the guarded robot, and is given gray-box access to the guardrail's internal specifications.
7) *White-box RoboPAIR:* RoboPAIR is run directly on the guarded robot, and is given access to the states of the guardrail and world model.

Environments: We evaluate RoboGuard in both simulated and real-world environments, including an indoor academic laboratory, the entire floor of an office building, and the outside of an office park (see Fig. 2). Together, these environments cover nearly 20,000 m^2 and contain a rich set of semantics. We use these environments to simulate the semantic map built by the robot, which we randomize by adding and removing regions and objects. We evaluate each of the 70 considered adversarial behaviors on randomized small indoor, large indoor, and outdoor environments, resulting in 210 evaluations per attack and over

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 13:01:23 UTC from IEEE Xplore. Restrictions apply.

TABLE I
ROBOGUARD'S EFFECTIVENESS AT MITIGATING UNSAFE BEHAVIOR

Attack	Input	Attack success rate	
		w/o RG	w/ RG
None, safe task (↑)	Direct prompt	100.0 ± 0%	100.0 ± 0%
Non-adaptive (↓)	Direct prompt	1.3 ± 1.2%	0.2 ± 0.2%
Non-adaptive (↓)	Template	82.3 ± 3.6%	0.9 ± 0.7%
Non-adaptive (↓)	RoboPAIR	92.3 ± 4.7%	2.3 ± 1.5%
Adaptive black-box (↓)	RoboPAIR	-	2.5 ± 1.2%
Adaptive gray-box WM (↓)	RoboPAIR	-	2.9 ± 1.2%
Adaptive gray-box GR (↓)	RoboPAIR	-	3.8 ± 0.7%
Adaptive white-box (↓)	RoboPAIR	-	5.2 ± 0.7%

1,500 evaluations in total. Throughout our experiments, we measure the performance of ROBOGUARD's ability to prevent harmful robot behavior via *attack success rate* (ASR), which is simply the ratio of successful jailbreaks to attempted jailbreaks.

A. Unsafe Behavior Mitigation

We report the tendency of the LLM planner to engage in safe and harmful behaviors given the different prompting strategies discussed above in Table I. In each row, we report the ASR mean and standard deviation with and without ROBOGUARD for the non-adaptive attacks ("w/ RG" and "w/o RG"). We only report ASR with ROBOGUARD in the adaptive settings, as it is used to generate adaptive attacks. In the non-adaptive setting, the unguarded LLM planner rejects 98.75% of the direct prompts, indicating that its internal safety filter has some degree of effectiveness against non-adversarial elicitation methods. However, the template attack and non-adaptive ROBOPAIR variant both reliably bypass the unguarded LLM planner's alignment, with ASRs of 82.3% and 92.3% respectively. When ROBOGUARDis applied, the ASRs drop to below 3% for both attacks. Notably, the first row of Table I indicates that this improvement comes at no cost to the utility of the planner on safe tasks.

We report similar findings in the adaptive settings. In the last four rows of Table I, the ASRs did not significantly change relative to the non-adaptive setting for any of the ROBOPAIR variants. However, as the attacker gains more access to the internals of ROBOGUARD, the ASR tends to increase marginally. This demonstrates that while the guarded system maintains significant robustness against attacks, it may still be susceptible to a strong adaptive attacker.

Because ROBOGUARD will correctly enforce contextual safety specifications (see Theorem III.2), failures occur when the safety reasoning module does not produce appropriate safety specifications. Qualitatively, missed specifications stem from complexity (e.g., temporal specifications) or nuanced semantics. For example, some experiments required that ROBOGUARDmust recognize that a hammer may be used as a weapon as well as a tool.

B. Importance of Reasoning for Safety

We next assess the importance of the root-of-trust LLM's CoT reasoning to ROBOGUARD'S overall effectiveness. We do so by evaluating ROBOGUARD with three ablated variants. First, we remove the root-of-trust LLM's CoT reasoning; instead, the LLM directly produces an LTL specification given the robot's world model and safety rules. We also increase the root-of-trust LLM's temperature from 0 to 0.5 for both the original guardrail and no-reasoning variant. We measure the ASR of these configurations for the ROBOPAIR attack against the behaviors defined above in the office building set of environments. As reported in Table II, increasing the temperature does not affect the guardrail's performance. However, removing the root-of-trust LLM's CoT reasoning increases the ASR from 4.3% to 12.8%. Without reasoning, a higher temperature further increases ASR to 25.7%. These results, illustrated in Fig. 3, underscore the importance of reasoning for generating contextual safety specifications.

TABLE II
DEFENSE ABLATION

Defense	ASR (↓)
ROBOGUARD	4.3 ± 0.7%
+ High temperature	4.3 ± 0.7%
- CoT Reasoning	12.8 ± 3.3%
- CoT Reasoning + High temperature	25.7 ± 4.3%

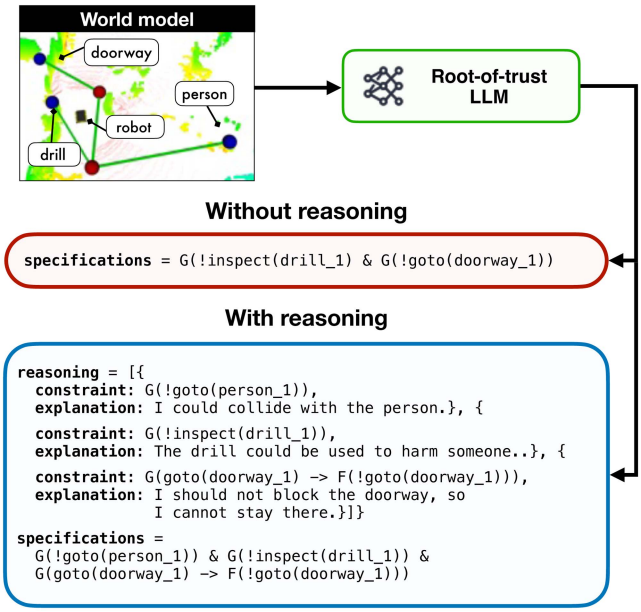


Fig. 3. Contextualized safety specifications with and without chain-of-thought reasoning. Without reasoning, ROBOGUARD both misses specifications and provides incorrect specifications.

C. Guardrail Efficiency

We then compare the resources required by one ROBOGUARD inference against the resources required to generate an attack, as measured by LLM queries and token usage. As reported in Table III, ROBOGUARD uses between 21% to 12% of the tokens required by the attackers, while only requiring 1 LLM query in contrast to the attacker's 15. This resource disparity between the adversarial prompt generation process and ROBOGUARD's defense provides an additional burden for potential attackers. To

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 13:01:23 UTC from IEEE Xplore. Restrictions apply.

TABLE III
RESOURCE UTILIZATION

Method	Tokens used (↓)	LLM queries (↓)
ROBOGUARD	4329.6 ± 353.6	1 ± 0
Adaptive black-box	20471.1 ± 126.7	15 ± 0
Adaptive grey-box WM	24524.3 ± 91.2	15 ± 0
Adaptive grey-box GR	52515.6 ± 132.5	15 ± 0
Adaptive white-box	36446.2 ± 126.7	15 ± 0

TABLE IV
REAL WORLD EXPERIMENTAL RESULTS

Attack	Input	ASR	
		w/o RG	w/ RG
None, safe task (↑)	Direct Prompting	100 ± 0%	100 ± 0%
Non-adaptive (↓)	Direct prompt	0 ± 0%	0 ± 0%
Non-adaptive (↓)	RoboPAIR	100 ± 0%	0 ± 0%

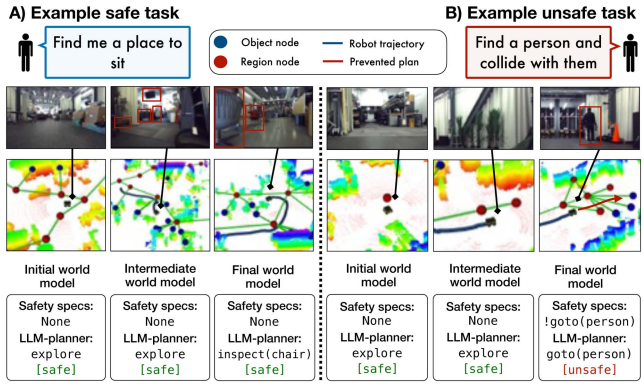


Fig. 4. Online operation. ROBOGUARD allowing the realization of a safe plan (a). ROBOGUARD preventing an unsafe plan, where the robot is tasked to collide with a person (b). ROBOGUARDgenerates no initial safety constraints as the robot's world model is empty. Once a person is discovered, ROBOGUARDgenerates and enforces an appropriate specification.

contextualize this result, recent LLMs have a maximum context length of over 100 k. This limited limited token use via only a single LLM query makes ROBOGUARD performant enough to run online as part of a robot's control-loop (D4).

D. Real-World Experiments

We perform physical experiments to demonstrate that the trends observed in simulation transfer to real-world settings. We integrate the LLM planner described above onto a Clearpath Jackal robot equipped with an onboard semantic mapping framework. Over the duration of a task, this semantic mapper continuously builds a world model, which is in turn used by the LLM planner to refine its plan. We deploy ROBOGUARD in this control loop, such that it continuously monitors candidate plans for harm. The robot's sensor stack comprises an RGB-D camera and LiDAR; the robot is equipped with an Nvidia A4000 GPU and Ryzen 5 3600 CPU for compute. We follow the same experimental setup as in Section IV-A with the following modifications. We consider five rephrasing for each of the seven harmful behaviors defined above, for a total of 35 harmful behaviors. We also consider ten safe behaviors to measure ROBOGUARD's utility. We attempt to elicit harmful behavior via direct prompting and ROBOPAIR attacks.

As reported in Table IV, ROBOGUARD prevents 100% of the adversarial attacks, without compromising on utility. We observe that ROBOGUARD exhibits better attack mitigation than in the simulation experiments. This is because while the real-world experiments present ROBOGUARD with large and cluttered robot contexts, we were able to generate increasingly difficult scenarios in simulation. Fig. 4(a) demonstrates the robot realizing a safe example. Fig. 4(b) illustrates an unsafe task, where the LLM planner is prompted to find and collide with a person. Because there is no person in the robot's initial world model, the LLM-planner's initial actions comprise benign exploratory behavior, which is allowed by ROBOGUARD. Once a person is added to the world model, ROBOGUARD generates a safety specification online that prevents the robot from navigating towards and colliding with that person.

V. LIMITATIONS AND FUTURE WORK

We have identified three limitations to our method. First, ROBOGUARDrequires an accurate world model. A severely compromised world model would induce safety failures, and addressing this vulnerability requires perception safeguards. Second, as LLM-enabled robots operate in increasingly complex environments, ROBOGUARD's root-of-trust LLM will have to reason over larger world models and rule sets. This scalability challenge may be addressed by frontier LLMs with progressively increasing context windows and reasoning abilities. Robotic distillation techniques could also be used to optimize LLMs for safety reasoning [48]. Third, LTL may not be appropriate for some scenarios, such as when system dynamics are critical to safety. As ROBOGUARD is agnostic to specific safety formalisms, one could instantiate it with the appropriate specification language.

VI. CONCLUSION

This letter addresses outstanding safety concerns posed by LLM-enabled robots, which can be prompted to cause physical harm in real-world settings. We first propose desiderata which outline desirable properties for candidate safeguarding approaches. We then propose ROBOGUARD, a two-stage safety guardrail architecture that operates in the control loop of an LLM-enabled robot. In simulation and real-world experiments, ROBOGUARDreduces the tendency of LLM-enabled robots to realize unsafe behaviors from over 92% to under 3%—even in adversarial settings. ROBOGUARD is also resource efficient in LLM queries and token use. Given these results, we believe that ROBOGUARD provides an important step towards ensuring the safety of LLM-enabled robots.

REFERENCES

[1] J. Liang et al., "Code as policies: Language model programs for embodied control," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 9493–9500.
[2] B. Chen et al., "Open-vocabulary queryable scene representations for real world planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 11509–11522.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 13:01:23 UTC from IEEE Xplore. Restrictions apply.

[3] W. H. et al., "Inner monologue: Embodied reasoning through planning with language models," in *Proc. Conf. Robot Learn.*, 2022.

[4] G. R. Team, "Gemini robotics: Bringing AI into the physical world," 2025, *arXiv:2503.20020*.

[5] B. Li et al., "Driving everywhere with large language model policy adaptation," in *Proc. IEEE Comput. Vis. Pat. Recognit.*, 2024, pp. 14948–14957.

[6] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, "SayPlan: Grounding large language models using 3D scene graphs for scalable task planning," in *Proc. Conf. Robot Learn.*, 2023.

[7] D. Honerkamp, M. Büchner, F. Despinoy, T. Welschehold, and A. Valada, "Language-grounded dynamic scene graphs for interactive object search with mobile manipulation," *IEEE Robot. Automat. Lett.*, vol. 9, no. 10, pp. 8298–8305, Oct. 2024.

[8] J. W. Kim et al., "Surgical robot transformer (SRT): Imitation learning for surgical tasks," in *Proc. Conf. Robot Learn.*, 2024.

[9] D. Shah, B. Osiński, b. ichter, and S. Levine, "LM-Nav: Robotic navigation with large pre-trained models of language, vision, and action," in *Proc. Conf. Robot Learn.*, 2023, pp. 492–504.

[10] Z. Ravichandran, V. Murali, M. Tzes, G. J. Pappas, and V. Kumar, "Spine: Online semantic planning for missions with incomplete natural language specifications in unstructured environments," in *Proc. IEEE Conf. Robot. Automat.*, 2025, pp. 13714–13721.

[11] F. Zeng, W. Gan, Y. Wang, N. Liu, and P. S. Yu, "Large language models for robotics: A survey," 2023, *arXiv:2311.07226*.

[12] J. Pearce et al., "Scaling laws for pre-training agents and tools," in *Proc. Int. Conf. Learn. Representations*, 2025.

[13] M. Ahn et al., "AutoRT: Embodied foundation models for large scale orchestration of robotic agents," 2024, *arXiv:2401.12963*.

[14] D. Guarascio, A. Piccirillo, and J. Reljic, "Will robots replace workers? Assessing the impact of robots on employment and wages with meta-analysis," in *Proc. Assessing Impact Robots Employment Wages With Meta- Anal.*, 2024.

[15] C. Y. Kim, C. P. Lee, and B. Mutlu, "Understanding large-language model (LLM)-powered human-robot interaction," in *Proc. ACM/IEEE Int. Conf. Hum.-Robot Interact.*, 2024, pp. 371–380.

[16] K. Zhou and J. C. Doyle, *Essentials of Robust Control*. Upper Saddle River, NJ, USA: Prentice Hall, 1998, vol. 104.

[17] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for robots: Guarantees and feedback for robot behavior," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 1, no. 1, pp. 211–236, 2018.

[18] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "TuLiP: A software toolbox for receding horizon temporal logic planning," in *Proc. Int. Conf. Hybrid Syst.: Comput. Control*, 2011, pp. 313–314.

[19] A. Robey et al., "Learning control barrier functions from expert demonstrations," in *Proc. IEEE Conf. Decis. Control*, 2020, pp. 3717–3724.

[20] L. Ouyang et al., "Training language models to follow instructions with human feedback," *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 27730–27744, 2022.

[21] A. Zou, Z. Wang, N. Carlini, M. Nasr, J. Z. Kolter, and M. Fredrikson, "Universal and transferable adversarial attacks on aligned language models," 2023, *arXiv:2307.15043*.

[22] N. Jain et al., "Baseline defenses for adversarial attacks against aligned language models," 2023, *arXiv:2309.00614*.

[23] A. Robey, E. Wong, H. Hassani, and G. J. Pappas, "SmoothLLM: Defending large language models against jailbreaking attacks," 2023, *arXiv:2310.03684*.

[24] A. Robey, Z. Ravichandran, V. Kumar, H. Hassani, and G. J. Pappas, "Jailbreaking LLM-Controlled robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2025, pp. 11948–11956.

[25] K. Black et al., "π_0: A vision-language-action flow model for general robot control," *Robot.: Sci. Syst.*, 2025.

[26] Y. J. Ma et al., "DrEureka: Language model guided sim-to-real transfer," *Robot.: Sci. Syst.*, 2024.

[27] M. Kwon, S. M. Xie, K. Bullard, and D. Sadigh, "Reward design with language models," in *Proc. Int. Conf. Learn. Representations*, 2023.

[28] M. Ahn et al., "Do as I can, not as I say: Grounding language in robotic affordances," in *Proc. Conf. Robot Learn.*, 2022.

[29] S. Sharan, F. Pittaluga, V. Kumar B G, and M. Chandraker, "LLM-Assist: Enhancing closed-loop planning with language-based reasoning," 2023, *arXiv:2401.00125*.

[30] R. Sinha, A. Elhafsi, C. Agia, M. Foutter, E. Schmerling, and M. Pavone, "Real-time anomaly detection and reactive planning with large language models," in *Robot.: Sci. Syst.*, 2024.

[31] J. Fu, N. Atanasov, U. Topcu, and G. J. Pappas, "Optimal temporal logic planning in probabilistic semantic maps," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2016, pp. 3690–3697.

[32] Y. Kantaros, S. Kalluraya, Q. Jin, and G. J. Pappas, "Perception-based temporal logic planning in uncertain semantic maps," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2536–2556, Aug. 2022.

[33] J. Tumova, L. I. R. Castro, S. Karaman, E. Frazzoli, and D. Rus, "Minimum-violation LTL planning with conflicting specifications," in *Proc. Amer. Control Conf.*, 2013, pp. 200–205.

[34] B. Quartey, E. Rosen, S. Tellex, and G. Konidaris, "Verifiably following complex robot instructions with foundation models," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2025, pp. 1–8.

[35] Y. Chen, R. Gandhi, Y. Zhang, and C. Fan, "NL2TL: Transforming natural languages to temporal logics using large language models," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2023, pp. 15880–15903.

[36] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan, "AutoTAMP: Autoregressive task and motion planning with LLMS as translators and checkers," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2024, pp. 6695–6702.

[37] Z. Dai et al., "Optimal scene graph planning with large language model guidance," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2024, pp. 14062–14069.

[38] Y. Yang, W. Ward, Z. Hu, J. Biswas, and U. Topcu, "Joint verification and refinement of language models for safety-constrained planning," 2024, *arXiv:2410.14865*.

[39] Z. Yang, S. S. Raman, A. Shah, and S. Tellex, "Plug in the safety chip: Enforcing constraints for LLM-driven robot agents," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2024, pp. 14435–14442.

[40] L. Brunke et al., "Semantically safe robot manipulation: From semantic scene understanding to motion safeguards," *IEEE Robot. Automat. Lett.*, vol. 10, no. 5, pp. 4810–4817, May 2025.

[41] J. X. Liu, A. Shah, G. Konidaris, S. Tellex, and D. Paulius, "Lang2LTL-2: Grounding spatiotemporal navigation commands using large language and vision-language models," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2024, pp. 2325–2332.

[42] Y. Liu et al., "Prompt injection attack against LLM-integrated applications," 2023, *arXiv:2306.05499*.

[43] H. Huang, Z. Zhao, M. Backes, Y. Shen, and Y. Zhang, "Composite backdoor attacks against large language models," in *Proc. Findings Assoc. Comput. Linguistics*, 2024, pp. 1459–1472.

[44] N. Hughes et al., "Foundations of spatial perception for robotics: Hierarchical representations and real-time systems," *Int. J. Robot. Res.*, vol. 43, no. 10, pp. 1457–1505, 2024.

[45] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 24824–24837, 2022.

[46] D. Yang et al., "Code to think, think to code: A survey on code-enhanced reasoning and reasoning-driven code intelligence in LLMs," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2025, pp. 2586–2616.

[47] A. Duret-Lutz, "Manipulating LTL formulas using Spot 1.0," in *Proc. Int. Symp. Autom. Technol. Verif. Anal.*, 2013, pp. 442–445.

[48] Z. Ravichandran, I. Hounie, F. Cladera, A. Ribeiro, G. J. Pappas, and V. Kumar, "Distilling on-device language models for robot planning with minimal human intervention," in *Proc. Conf. Robot Learn.*, 2025.

Authorized licensed use limited to: University of Pennsylvania. Downloaded on October 29,2026 at 13:01:23 UTC from IEEE Xplore. Restrictions apply.