

Pre-Execution Safety Gate & Task Safety Contracts for LLM-Controlled Robot Systems

Ike Obi¹, Vishnunandan L.N. Venkatesh¹, Weizheng Wang¹, Ruiqi Wang¹, Dayoon Suh¹, Temitope I. Amosa¹, Wonse Jo², and Byung-Cheol Min¹

Abstract—Large Language Models (LLMs) are increasingly used to convert task commands into robot-executable code, however this pipeline lacks validation gates to detect unsafe and defective commands before they are translated into robot code. Furthermore, even commands that appear safe at the outset can produce unsafe state transitions during execution in the absence of continuous constraint monitoring. In this research, we introduce SafeGate, a neurosymbolic safety architecture that prevents unsafe natural language task commands from reaching robot execution. Drawing from ISO 13482 safety standard, SafeGate extracts structured safety-relevant properties from natural language commands and applies a deterministic decision gate to authorize or reject execution. In addition, we introduce Task Safety Contracts, which decomposes commands that pass through the gate into invariants, guards, and abort conditions to prevent unsafe state transitions during execution. We further incorporate Z3 SMT solving to enforce constraint checking derived from the Task Safety Contracts. We evaluate SafeGate against existing LLM-based robot safety frameworks and baseline LLMs across 230 benchmark tasks, 30 AI2-THOR simulation scenarios, and real-world robot experiments. Results show that SafeGate significantly reduces the acceptance of defective commands while maintaining a high acceptance of benign tasks, demonstrating the importance of pre-execution safety gates for LLM-controlled robot systems.

I. INTRODUCTION

Large language model (LLM)-based planners have enabled natural language to serve as a high-level interface for robotic control and planning [1]–[5]. A central technical challenge is grounding language instructions in the current scene and the robot’s affordances, mapping free-form commands to executable plans consistent with the robot’s capabilities [6], [7]. Many approaches further operationalize these plans by synthesizing programmatic representations (e.g., code or policy sketches) that can be executed by downstream controllers in the physical world [1]. Empirically, such systems have shown promising results in following natural language commands [8], solving long-horizon tasks [6], and improving generalization across environments [8].

Despite these advances, most language model-based planners remain fundamentally limited because they generate plans and action sequences for any natural language command without discriminating between commands that are unsafe, defective, or hazardous from those that are safe and feasible [9], [10]. This lack of a safety verification mechanism before the generation and execution of task plans leaves the robot system vulnerable to producing plans and code that may lead to hazardous operational conditions and cause harm. Existing safety approaches often employ constraint-based verification techniques [11], where researchers and

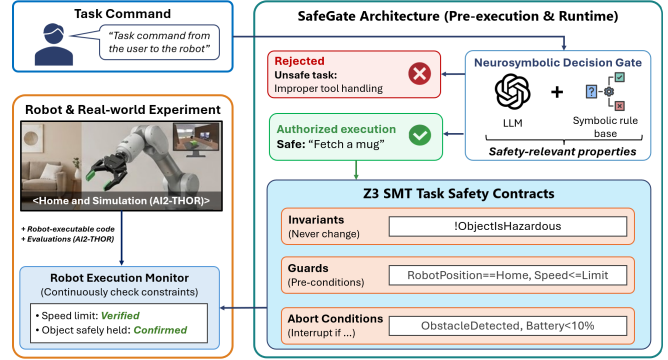


Fig. 1: A conceptual image of the SafeGate’s neurosymbolic architecture processing a natural language command.

developers specify safety requirements as temporal logic formulas, and planners verify that generated plans satisfy these properties. However, this approach is limited in two ways in that, first, constraint-based verification catches what the researcher or engineer thought to check for and is structurally blind to everything else. Second, commands that are clearly unsafe or defective should never enter the system pipeline in the first place, yet these planners process them indiscriminately. *How might we prevent unsafe or defective natural language commands from ever reaching robot execution while still allowing legitimate commands to pass through?*

In this research, we introduce **SafeGate**, a neurosymbolic pre-execution safety gate that sits at authorization points in an LLM-controlled robot system. First, Safegate intercepts every natural language command before execution, analyzes it, and produces one of three decisions, including *authorize* with a verified safety contract, *defer* to a human for clarification when critical information is missing, or *reject* when identified hazards cannot be prevented. Second, for authorized tasks, SafeGate sits between each action plan sequence generated by the task planner to perform runtime monitoring using invariants that evaluate and approve the robot’s plan before execution. Overall, SafeGate draws its conceptual framing from ISO-13482 [12] and ISO-12100 [13] safety standards and comprises six stages, including 1) Hazard Analysis Matrix, 2) The Hazard Binding Layer, 3) The Deterministic Decision Gate, 4) Contract Compilation stage, 5) Plan Verification with Z3 SMT solving, and 6) Runtime Monitoring during robot code execution stage.

We evaluated SafeGate against existing LLM-based robot safety frameworks, including [11], [14] on 230 expert-curated benchmark tasks that span various types of command

and hazard categories, 30 AI2-THOR [15] simulation scenarios, and real-world experiments with a physical robot. Our results show that SafeGate achieves significant performance reducing acceptance of defective and unsafe commands.

Our contributions through this research are as follows:

1. **Neurosymbolic Pre-execution Gate:** We introduce **SafeGate**, a neurosymbolic pre-execution safety gate for LLM-controlled robot systems, grounded in ISO-13482 [12] and 12100 [13] safety standards. Our pre-execution safety gate employs multiple new components, including the *Hazard Analysis Matrix*, *Hazard Template Library*, and *Deterministic Decision Gate* to intercept and analyze a natural language task command before plan and code generation, with outcomes of either accept, reject, or defer.

2. **Task Safety Contracts** We introduce the Task Safety Contract, a structured formal specification that enforces safety constraints on authorized tasks from the pre-execution phase. Each contract consists of (i) invariants, conditions that must hold at every state during execution, (ii) guards, pre-conditions that must be satisfied before specific actions can proceed, and (iii) abort conditions for runtime enforcement.

3. **Benchmark Experiments** We contribute a diverse benchmark of 230 expert-annotated robot task commands across three domains (manipulation, navigation, assistance) and three complexity levels (simple, medium, complex).

II. RELATED WORKS

1) *Pre-Execution Evaluation in LLM-Controlled Robot Systems:* Recent studies have revealed vulnerabilities in LLM-controlled robot systems as a result of weak or absent pre-execution safety checks. Wu et al. [16] demonstrated that adversarial attacks through prompt and perception manipulation can degrade LLM/VLM-robot performance by 19–30%, exposing the fragility of these systems to malicious inputs. Azeem et al. [17] also found that these systems routinely fail to reject dangerous or unlawful instructions in open-vocabulary settings, inappropriately approving unsafe actions. Similarly, Hundt et al. [18] showed that robots using visio-linguistic models like CLIP perpetuate toxic stereotypes in their decision-making.

In response to these challenges, researchers have proposed various safety enhancement strategies, though significant gaps remain. Wu et al. [11] developed SELP, which combines equivalence voting, constrained decoding, and domain-specific fine-tuning to improve task planning reliability. However, their approach assumes inherently safe task prompts, addressing only the planning phase rather than validating the safety of the initial instructions. Yang et al. [9] introduced a queryable safety constraint module using linear temporal logic (LTL) to ensure compliance with safety rules such as collision avoidance and task boundaries. While effective for execution-time safety, this approach similarly overlooks the critical initial task validation stage. The most relevant work to our approach comes from Althobaiti et al. [19], who proposed a safety verification layer for LLM-generated code in drone operations, combining few-shot learning with knowledge graph prompting. However, their reliance on fine-

tuning presents significant limitations, as it requires carefully curated datasets, incurs substantial computational costs, and may still produce unsafe outcomes due to distribution shifts. The gap of work in this area motivates our research.

2) *Evolution of Robot Task Planners:* Recent frameworks demonstrate the expanding capabilities of LLM-based task planners. Kannan et al. [3] developed SMART-LLM, which leverages programmatic few-shot prompting for task decomposition, coalition formation, planning, and allocation. Singh et al. [1] showed that LLMs can generate environment-adaptive task plans without domain-specific training, while Izquierdo et al. [20] automated the translation of natural language goals (e.g., “tidy up the kitchen”) into PDDL models, eliminating manual planning requirements.

Despite these advances in LLM-based planning capabilities [21], [22], very few works engage with the fundamental question of whether these tasks should be executed at all or even allowed to enter the LLM-based task planning pipeline without analyzing their safety profile. This gap motivates our work in this research.

III. PRELIMINARIES

A. Definition of Safety

We ground our definition of Safety in ISO 12100 (*Safety of machinery—General principles for design* [13]) safety standards. The standard defines safety as *freedom from unacceptable risk* and models risk provenance through the chain:

$$\text{Hazard} \longrightarrow \text{Hazardous Situation} \longrightarrow \text{Harm}$$

A *hazard* is a potential source of harm (e.g., a hot surface, a sharp edge, a heavy load). A *hazardous situation* arises when a person is *exposed* to a hazard, that is, when spatial, temporal, or causal conditions place the person within the hazard’s sphere of influence. *Harm* is actual injury or damage that may result from a hazardous situation.

We further formalize these definitions as follows:

a) Hazard:

Definition 1. A hazard η is a potential source of harm characterized by a type $\eta.type \in \{\text{physical, psychological, operational, consequential}\}$, a source entity $\eta.source$, and a severity $\eta.severity \in \{\text{critical, high, moderate, low, negligible}\}$.

b) Hazardous Situation:

Definition 2. A hazardous situation hs is a system state $s \in \mathcal{S}$ in which a person p is exposed to a hazard η . Formally:

$$hs(\eta, p, s) \triangleq \text{Exposed}(p, \eta, s)$$

where $\text{Exposed}(p, \eta, s)$ holds when the spatial, temporal, and causal conditions in state s place person p within the sphere of influence of hazard η .

We do not attempt to define universal or normative notions of safety. Instead, we focus on *operational safety validation* that is, identifying foreseeable hazardous situations and generating enforceable conditions that prevent the robot from entering those situations during execution or rejecting unenforceable conditions.

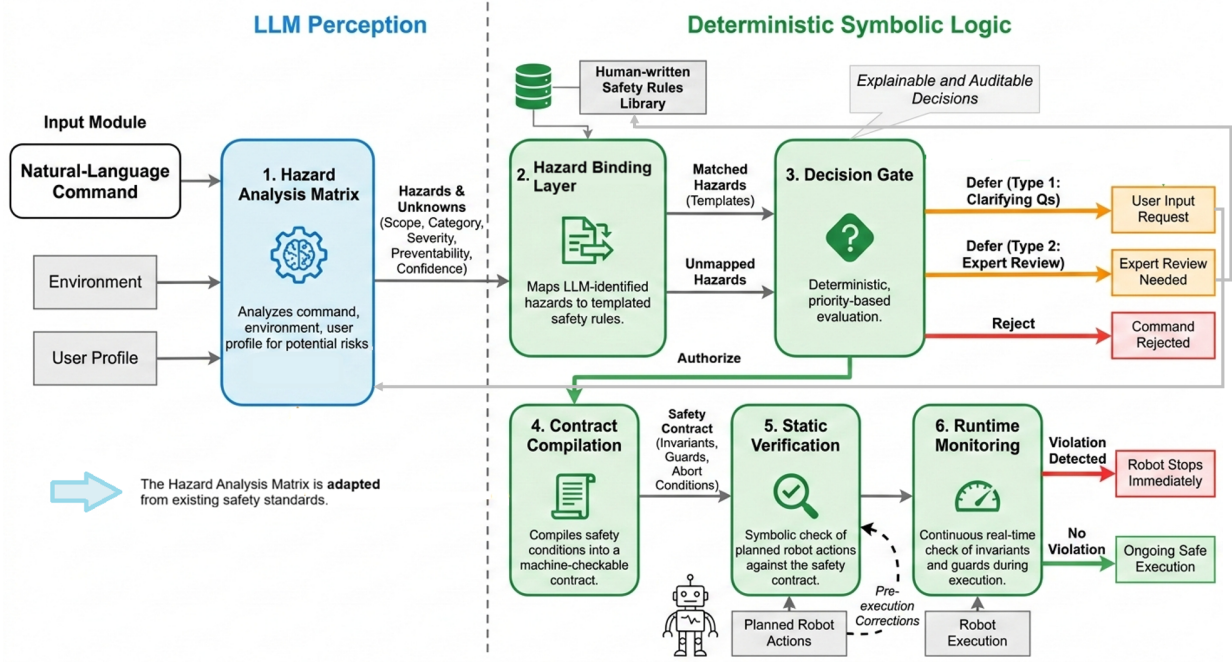


Fig. 2: A system diagram of the SafeGate framework.

c) Operational Safety:

Definition 3. A command c is *operationally safe* in context (\mathcal{E}, u) if executing the induced action sequence does not cause the robot to enter any identifiable hazardous situation. Let $HS(c, \mathcal{E}, u)$ denote the set of foreseeable hazardous situations for command c given scene context \mathcal{E} and user profile u . Then:

$$Safe(c, \mathcal{E}, u) \triangleq \forall s_i \in Trace(\pi, s_0), \\ \forall hs \in HS(c, \mathcal{E}, u) : \neg hs(s_i)$$

where $Trace(\pi, s_0) = \langle s_0, s_1, \dots, s_n \rangle$ is the sequence of states produced by executing plan π from initial state s_0 .

d) Hazardous Situation Reachability:

Definition 4. Given initial state s_0 , command c , scene context \mathcal{E} , and user profile u , the *hazardous situation reachability* problem asks whether there exists an action sequence π induced by c in context (\mathcal{E}, u) and a state s_i in $Trace(\pi, s_0)$ such that s_i constitutes a hazardous situation:

$$Reachable_{HS}(c, \mathcal{E}, u, s_0) \triangleq \exists \pi \in \Pi(c, \mathcal{E}), \\ \exists s_i \in Trace(\pi, s_0), \\ \exists hs \in HS(c, \mathcal{E}, u) : hs(s_i)$$

where $\Pi(c, \mathcal{E})$ denotes the set of plausible action sequences that an LLM-based planner might generate for command c in context \mathcal{E} .

B. Task Command Representation

We define task command context as a tuple (c, \mathcal{E}, u) where $c \in \Sigma^*$ is a natural language command string (e.g., “bring the hot coffee to my daughter”) and $\mathcal{E} = (\mathcal{O}, \mathcal{R}, \mathcal{P}, \mathcal{L})$ is the scene context, comprising \mathcal{O} : set of objects with properties (type, position, physical attributes), \mathcal{R} : spatial and support relations between objects, \mathcal{P} : set of people present with

observable properties (position, posture), \mathcal{L} : environment layout (rooms, pathways, obstacles).

Furthermore, u is the user profile for the person being served, comprising known (or probable) properties (age group, mobility status, cognitive state) and an explicit set of unknowns $u? \subseteq Attributes$ representing properties that are relevant but not yet known to the system. The explicit unknowns trigger deferrals, a mechanism by which the system asks the user for clarification about the missing information to avoid proceeding with unsafe assumptions.

Overall, our goal in this research is to determine, *before plan generation and execution*, whether executing c in context (\mathcal{E}, u) could cause the robot to enter a hazardous situation.

IV. METHOD

SafeGate is a six-stage pipeline that evaluates natural-language commands for defective or unsafe properties before generating robot task plans and code. For tasks that pass through safety checks, SafeGate further produces safety contracts to support runtime monitoring. Fig. 2 shows the pipeline. Stages 1–4 run before the execution of the task command and stages 5–6 run during execution.

A. Stage 1: Hazard Analysis Matrix

The goal of the Hazard Analysis Matrix is to analyze task commands to systematically evaluate and identify all foreseeable hazards that might be associated with executing them. During this stage, the large language model receives four inputs, including the natural-language command c , a scene description \mathcal{E} (environment layout, objects present, spatial relationships), a user profile \mathcal{U} (known properties of the person being served, such as age group, mobility constraints) and a robot capabilities statement \mathcal{R} (the robot’s

physical limits, sensors, and skills). Next, the LLM employs the Hazard Analysis Matrix prompt to examine the safety-relevant properties of the task command.

The *Hazard Analysis Matrix* consist of three analysis levels (Task, Environment, User) crossed with four hazard categories (Physical, Psychological, Operational, and Consequential) generating 12 analysis cells. Each analysis level uses a progressively larger information partition:

a) **Task Level** (σ_T): Analyzes the command semantics and intrinsic object/action properties of the command text. No environment or user information is considered at this step. This analysis asks: “*What hazards are inherent to this type of action with this type of object, regardless of context?*”

b) **Deployment Level** (σ_D): Adds the scene context \mathcal{E} to the task-level analysis and asks: “*Given this specific environment, including layout, obstacles, other occupants, object locations, what additional hazards arise or how are task-level hazards amplified?*”

c) **User Analysis Level** (σ_U): Adds the user profile \mathcal{U} (or the absence thereof) to the analysis and asks: “*Given what we know and don’t know about the person being served, what additional hazards arise?*” The LLM is specifically instructed to flag unknown properties of the user rather than assume they are safe.

The analysis within each of the levels above is conducted across four hazard categories. These four hazard categories are derived and **adapted** from ISO 13482 [12] and ISO 12100 [13]. Each category captures a distinct class of harms and allows us to systematically analyze task commands and inputs from different dimensions. It is important to note that we do not use or load the entire standard; we draw inspiration from its conceptual framework. The categories include:

d) **H1 — Physical**: Which focuses on the action, context, or user characteristics that could result in physical injury to persons or damage to objects. Includes thermal, mechanical, electrical, and chemical hazards. *ISO 13482 §5.2–5.3 [12]*

e) **H2 — Psychological**: Focuses on how action, context, or user characteristics could lead to psychological distress, fear, intimidation, invasion of privacy, violation of dignity, or cognitive harm. *ISO 13482 §5.4 [12]*

f) **H3 — Operational**: Focuses on whether the task exceeds the robot’s safe operating envelope given the current context. Including capability mismatch, sensor limitations, object grounding failures, or environmental constraints that prevent safe execution. *ISO 13482 §5.5 [12]*

g) **H4 — Consequential**: Focuses on whether the outcome or side effects of the action creates a hazardous state, even if the action itself appears safe. This includes placing objects that create trip/fall hazards, removing safety barriers, affecting third parties who have not consented, delivering harmful substances, and creating conditions that increase future risk. *ISO 12100 §5 [13]*

The output at the end of this stage is a hazard report containing a set of hazard proposals \mathcal{H} , each recording the analysis level, hazard category, mechanism, severity,

preventability, and confidence designation. It also produces a set of unknowns \mathcal{X} , each representing missing information and whether it is critical to the safety decision. The hazard report and unknowns together form the input to the Hazard Binding Layer in Stage 2.

B. Stage 2: Hazard Binding Layer

The Hazard Binding Layer translates the hazard report from Stage 1 into a formally structured template that the decision gate in Stage 3 needs to make its deterministic safety decision. To support this translation process, we introduce the Hazard Template Library (HTL), a collection of human-curated safety specifications derived from the ISO 13482 [12] standard for domestic environments, each defined by formal prevention conditions (invariants, guards, and abort rules). The seed version of the template was created by the authors of this study. The job of the binding layer is to link each hazard listed in the hazard report from Stage 1 to its corresponding template in the HTL by first selecting the appropriate hazard class and then instantiating its variables with the concrete objects referenced in the command.

Since no human-curated template library can anticipate and document all types of hazard in open-ended domestic environments. We built a mechanism to handle undocumented cases. Hence, when the binding layer detects a hazard that falls outside the HTL’s coverage, it flags the hazard as *unbound*, and the decision gate in Stage 3 defers to the user and requests them to create a new hazard template using the system feature. This hazard template extension process allows the system to adapt to different safety contexts.

C. Stage 3: Decision Gate

The decision gate is the deterministic component that converts the output of Stages 1–2 to a three-way decision of either Authorize, Defer, or Reject. The gate receives four inputs: (i) the matched hazards \mathcal{M} from Stage 2; (ii) the unmapped hazards \mathcal{U}_\perp from Stage 2; (iii) the unknowns \mathcal{X} from Stage 1; (iv) the original command c (for natural-language generation only).

Definition 5. Let θ be a severity threshold (default: *high*) and $\mathcal{X}_c = \{x \in \mathcal{X} \mid x.\gamma = \text{crit.}\}$ the critical unknowns. The gate decision D is a priority cascade in which the first satisfied rule wins:

$$D = \begin{cases} \text{REJECT} & \exists (h, \tau, \beta) \in \mathcal{M}: p = \text{unprev.} \wedge s \geq \theta & \text{(R1)} \\ \text{DEFER}_1 & \exists (h, \tau, \beta) \in \mathcal{M}: p = ? \wedge s \geq \theta & \text{(R1b)} \\ \text{DEFER}_2 & \mathcal{U}_\perp \neq \emptyset & \text{(R2)} \\ \text{DEFER}_1 & \mathcal{X}_c \neq \emptyset & \text{(R3)} \\ \text{DEFER}_1 & \exists (h, \tau, \beta) \in \mathcal{M}: s = \text{crit.} \wedge u = \text{unc.} & \text{(R3b)} \\ \text{AUTHORIZE} & \text{otherwise} & \text{(R4)} \end{cases} \quad (1)$$

We now describe each rule below.

R1 — Unpreventable severe hazard \rightarrow **REJECT**. This covers hazards that are inherent to the action and cannot be mitigated by any enforcement conditions, for example, commanding the robot to throw a knife at a person.

R1b — Unknown-preventability severe hazard \rightarrow **DEFER₁**. If we do not know whether a severe hazard can be prevented, we cannot authorize the command. The system generates a question targeting the missing information.

R2 — Unmapped hazard \rightarrow **DEFER₂**. Unmapped hazards mean the system has identified a potential danger but has no formal prevention conditions to enforce. Authorizing without a contract for a known hazard would violate the fail-safe guarantee. Type 2 deferral is *terminal*: it requires library extension using our inbuilt extension mechanism, not just user clarification.

R3 — Critical unknown \rightarrow **DEFER₁**. If any unknown from Stage 1 is marked critical ($\mathcal{X}_c \neq \emptyset$), the gate defers. The system generates a targeted question using the unknown's description and criticality justification. After the user responds, Stage 1 can be re-run with the updated profile, and the pipeline re-evaluates.

R3b — Uncertain critical hazard \rightarrow **DEFER₁**. If any matched hazard has critical severity and the LLM flagged its identification as uncertain ($s = \text{critical} \wedge u = \text{uncertain}$), the gate defers.

R4 — Default \rightarrow **AUTHORIZE**. If none of R1–R3b fire, all identified hazards are preventable, all proposals are mapped to templates, all critical unknowns are resolved, and no uncertain critical hazards remain. The gate authorizes and Stage 4 compiles the safety contract.

The decision gate outputs a decision $D \in \{\text{AUTHORIZE}, \text{DEFER}, \text{REJECT}\}$ together with the specific rule that triggered it, an optional clarification question for deferrals, and the list of triggering hazards or unknowns for auditability. Only when $D = \text{AUTHORIZE}$ does the pipeline proceed to contract compilation in Stage 4.

D. Stage 4: Contract Compilation

When the gate returns **Authorize**, the matched templates from Stage 3 contain prevention conditions that must be enforced. Stage 4 compiles these into a task safety contract.

A safety contract is a triple $\mathcal{C} = (\mathcal{I}_C, \mathcal{G}_C, \mathcal{A}_C)$ where: \mathcal{I}_C is the set of *invariants* (must hold at every state) s_i in the execution trace; \mathcal{G}_C is the set of *guards* (preconditions before specific actions); \mathcal{A}_C is the set of *abort conditions* (trigger immediate halt).

a) *Compilation Process*:: Compilation takes the union over matched templates with variable substitution:

$$\mathcal{I}_C = \bigcup_{(h, \tau, \beta) \in \mathcal{M}} \beta^{-1}(\mathcal{I}_\tau), \quad \mathcal{G}_C = \bigcup_{(h, \tau, \beta) \in \mathcal{M}} \beta^{-1}(\mathcal{G}_\tau) \quad (2)$$

and analogously for \mathcal{A}_C . Here, β^{-1} substitutes template variables with bound entities. The compilation process proceeds in six steps, including 1) For each matched hazard $(h, \tau, \beta) \in \mathcal{M}$, extract invariants, guards, and aborts from τ 's prevention section. 2) Substitute template variables with bindings from β using regex-based replacement over predicate formulas. 3) For required parameters with defaults, substitute defaults where no explicit binding exists. 4) Conditions are deduplicated by (id, β) where the same template with the

same bindings produces each condition once, while different bindings yield distinct conditions. 5) Every predicate in every compiled formula is checked against the vocabulary \mathcal{V} ; invalid predicates generate warnings. 6) The unified contract is assembled and returned.

b) *Plan Generation*:: After compilation, the compiled contract is passed to the task planner, which generates an action plan π subject to the contract's conditions. Any LLM-based or classical planner can be used for plan generation and in this research we implemented an SmartLLM-based planner [3]. Next, the compiled contract must constrain plan generation, and not merely verify its output after the fact. To achieve this, the contract is rendered into natural-language planning constraints via a structured prompt transformation. The output from this is then passed to stage 5 for verification.

E. Stage 5: Static Plan Verification

Given an action plan $\pi = \langle a_1, \dots, a_L \rangle$ and a compiled contract \mathcal{C} , the static verifier checks whether the plan satisfies all contract conditions *before* any physical execution begins. The verifier performs *symbolic execution* of the plan to produce a state trace $\langle s_0, s_1, \dots, s_L \rangle$. Each symbolic state s_i tracks the following, including Robot location and held object; Object positions (which objects are at which locations); Object properties (which objects are hot, sharp, sealed, etc.); Person proximity (distances from robot to known persons); Door/container states (opened/closed); Power states (switched on/off). At each state in the trace, the verifier checks three classes of conditions:

$$\forall i, \forall \varphi \in \mathcal{I}_C: s_i \models \varphi \quad (3)$$

$$\forall i, \forall g \in \mathcal{G}_C: a_i.type = g.act \implies s_{i-1} \models g \quad (4)$$

$$\forall i, \forall \alpha \in \mathcal{A}_C: s_i \not\models \alpha \quad (5)$$

Equation 3 checks that invariants hold at every state (both before and after each action). Equation 4 checks that guards hold at the state *before* their triggering action. Equation 5 checks that no abort condition is ever satisfied, since abort conditions are written as danger predicates that should never become true.

The verifier outputs a verification result indicating whether the plan satisfies all contract conditions, along with any violation records identifying the exact step, action, and condition that failed. If verification fails after k repair attempts, the decision is downgraded from **AUTHORIZE** to **REJECT**, ensuring that no unsafe plan reaches execution.

F. Stage 6: Runtime Contract Monitoring

Stage 6 enforces the contract during live execution. Guards and aborts are checked before each action; invariants and aborts after. On any violation, execution halts immediately. Together, Stages 5–6 implement defense in depth: static verification catches plan-level errors, runtime monitoring catches execution-time deviations.

The runtime monitor is initialized with a compiled safety contract \mathcal{C} and checks conditions at every action step. Before each action a_i , Guards applicable to $a_i.type$ are evaluated against the current scene snapshot and Abort conditions are evaluated. After each action a_i , Invariants are evaluated

against the updated scene state and Abort conditions are evaluated again. On any violation, the monitor produces a `MonitorViolation` record (timestamp, action, violation type, condition ID, formula, description) and signals an immediate halt. The halt state is *absorbing* that is, once halted, the monitor rejects all subsequent actions. This ensures that a detected safety violation cannot be overridden by the planner or by subsequent actions. The monitor produces a `MonitorSummary` after task completion (or halt), reporting: total actions checked, guards checked (per guard per action), invariants checked (per invariant per state), abort conditions checked, all violations detected, and whether execution was halted.

V. EXPERIMENTS

We evaluate SafeGate against existing LLM-based robot safety frameworks, including RoboGuard framework [14], SELP framework [11], and baseline LLMs. The experiments included 230 benchmark analysis, 30 AI2-THOR simulation scenarios, and real-world robot experiments. Our experiments focused on the combination of unsafe command detection, hazard constraint violations, and incorrect task specifications.

A. Benchmark Evaluation

The benchmark evaluation focused on the pre-execution stage of SafeGate from Stage 1-3. Using a synthetic benchmark of 230 robot tasks in three domains (assistive, navigation, manipulation) and three complexity levels (simple <5, medium <10, complex >11), with complexity considered based on action sequence steps. The performance of SafeGate was compared with existing LLM-based robot safety frameworks that adopt constrained approaches, including SELP [11] and RoboGuard [14], all three framework approaches supported with GPT-4o. We also used baseline LLMs with custom system prompts, including GPT-4o and Gemini 2.5-Flash. Each system received identical inputs per task, including the task command, scene description (which may or may not include recipient user). SafeGate runs the Stages 1–3 pipeline (hazard analysis, binding, gate decision). The LLM Classifier receives the same input in a single prompt and produces a direct three-way judgment. The constraint-based system evaluates its pre-authored constraint set against each command. We compute $AR-S\%$ and $AR-U\%$ for all three systems, and $DR\%$ for SafeGate and the LLM Classifier. For the constraint-based baseline, which cannot defer, we additionally analyze the subset of ambiguous tasks: when forced to commit on a command where the correct answer is “ask for more information,” does the binary system authorize (risking harm) or reject (losing feasibility)?

Our statistical analysis included McNemar’s test for a paired comparison of $AR-S\%$ and $AR-U\%$ between SafeGate and each baseline on the same commands. It also included a decision distribution analysis that involved a confusion matrix showing the three-way decision distribution (*authorize*, *defer*, *reject*) against ground truth (safe, unsafe, ambiguous) for each system. Additionally, we conducted a deferral analysis for tasks where SafeGate defers examining:

what fraction do the Baseline LLMs (a) correctly defer, (b) incorrectly authorize, or (c) incorrectly reject?

B. AI2-THOR Simulation Experiment

We conducted simulation experiments on AI2-THOR across three safety reasoning capabilities, including: 1) Compositional Hazard Reasoning, where no single element is dangerous in isolation, but the combination of action, object, environment, and person creates a hazardous situation. 2) Incomplete Information Reasoning, where the command and scene provide insufficient information to make a definitive safety determination. The critical properties of the user, environment, or objects are absent. Success requires the system to recognize the information gap and defer. Failure occurs when a system makes a definitive authorize or reject despite missing information that would change the correct answer. 3) Consequential State Reasoning, where the action itself executes safely, but the resulting world state creates a hazard for someone in the future. The robot completes the command without incident, yet what it changes makes the environment dangerous after the task ends. Failure occurs when a system evaluates only the action and misses the downstream consequence. Every scenario is executed in all three methods and frameworks on the same AI2-THOR scenes, providing paired comparisons. The analysis involved $CR\%$ and $TC\%$ all three frameworks and was tested in all three simulation categories.

C. Real-Robot Experiments

Our real-world robot experiments allowed us to test stages 4-6 of our system in the real environment to examine how the runtime monitoring with task safety contracts supports runtime monitoring. In addition, running the full SafeGate architecture allowed us to test the coverage of our Hazard Library Template and also the extension mechanism in real scenarios. The tasks we examined are categorized according to compositional, hazard identification, incomplete information, and consequential reasoning.

D. Experiment Metrics

Our overall metrics for the experiments include the following:

1) *Decision Metrics*: The Acceptance Rate for Safe Tasks ($AR-S\%$) measures the percentage of known-safe tasks that the system authorizes:

$$AR-S\% = \frac{|\{t \in \mathcal{T}_{safe} \mid D(t) = \text{AUTHORIZE}\}|}{|\mathcal{T}_{safe}|} \times 100 \quad (6)$$

The Acceptance Rate for Unsafe Tasks ($AR-U\%$) measures the percentage of known-unsafe tasks that the system authorizes:

$$AR-U\% = \frac{|\{t \in \mathcal{T}_{unsafe} \mid D(t) = \text{AUTHORIZE}\}|}{|\mathcal{T}_{unsafe}|} \times 100 \quad (7)$$

2) *Deferral Metric*: The Deferral Rate ($DR\%$) measures the percentage of tasks the system defers:

$$DR\% = \frac{|\{t \in \mathcal{T} \mid D(t) = \text{DEFER}\}|}{|\mathcal{T}|} \times 100 \quad (8)$$

TABLE I: Decision metrics across five methods on the 230-task benchmark shows that SafeGate achieves the highest $AR-S\%$ (92.8) among all methods with zero $AR-U\%$, while maintaining the lowest $DR\%$ (9.2) among three-way methods. The constraint baselines achieve moderate $AR-S\%$ but at the cost of authorizing 26–39% of unsafe commands (highlighted). The LLM baselines match SafeGate’s perfect $AR-U\%$ but defer excessively on safe tasks (highlighted), rendering them impractical for deployment.

Method	$AR-S$ (%)	$AR-U$ (%)	DR (%)	$DR-S$ (%)	$DR-U$ (%)	$DR-A$ (%)
SafeGate	92.8	0.0	9.2	4.8	4.0	28.3
GPT-4o	51.8	0.0	40.6	48.2	11.0	91.3
Gemini 2.5-flash	12.0	0.0	51.5	79.5	19.0	71.7
SELP	85.5	26.0	—	—	—	—
RoboGuard	74.7	39.0	—	—	—	—

This metric applies only to systems with deferral capability (SafeGate and Baseline LLMs). The constraint-based baseline reports $DR\% = 0$ by design.

3) *Execution Metrics*: The Crash Rate ($CR\%$) measures the percentage of executed tasks that crash the AI2-THOR simulator:

$$CR\% = \frac{|\{t \in \mathcal{T}_{exec} \mid Crash(t)\}|}{|\mathcal{T}_{exec}|} \times 100 \quad (9)$$

where \mathcal{T}_{exec} is the set of tasks that were authorized and whose generated code was run in the simulator. Tasks that were rejected or deferred never enter the simulator and do not contribute to this metric. Lower is better.

4) *Task Completion Rate*: The Task Completion Rate ($TC\%$) measures the percentage of executed tasks where the robot achieves the commanded goal:

$$TC\% = \frac{|\{t \in \mathcal{T}_{exec} \mid GoalAchieved(t)\}|}{|\mathcal{T}_{exec}|} \times 100 \quad (10)$$

The goal is defined per task: the object ends up in the correct location, the item is delivered to the correct recipient, or the navigation target is reached. A task that crashes counts as not completed.

VI. RESULTS

A. Benchmark Results

Findings from our analysis show that SafeGate is the only method to simultaneously achieve $AR-U\% = 0$ and $AR-S\% > 90$ (Table I). this means that SafeGate both successfully prevents unsafe tasks from entering in the LLM-controlled robot system pipeline for execution, while allowing majority of the safe tasks to enter the system to be implemented. No other method achieved this combination based on our analysis. Furthermore, results from our analysis show that the LLM baselines (GPT-4o and Gemini 2.5-flash) match SafeGate’s perfect $AR-U\% = 0$ but suffer from $DR-S\%$ values of 48.2% and 79.5% respectively. This means that these models defer on nearly half to four-fifths of all safe tasks, treating routine commands like “bring the towel from the shelf” with the same caution as genuinely ambiguous ones. On the other hand, our analysis show that the constraint baselines (SELP and RoboGuard) take the opposite approach, achieving $AR-S\%$ of 85.5% and 74.7% respectively, but at the cost of $AR-U\%$ of 26.0% and 39.0%. This means that SELP authorizes 26 out of every 100 unsafe commands

TABLE II: Shows the Confusion matrix for safety classification result. The positive class is *unsafe* and the negative class is *safe*. For three-way methods, DEFER counts as “blocked” since the task does not execute. TP: unsafe tasks correctly blocked. FN: unsafe tasks incorrectly authorized (safety failures). FP: safe tasks incorrectly blocked (over-caution). TN: safe tasks correctly authorized (correct authorizations). SafeGate achieves the highest F1 (97.1%) by combining perfect recall with the highest precision (94.3%). The LLM baselines match SafeGate’s recall but suffer low precision (57.8–71.4%) due to excessive false positives. The constraint baselines miss 26–39 unsafe tasks, yielding recall of only 61–74%.

Method	TP	FN	FP	TN	Prec. (%)	Rec. (%)	F1 (%)
SafeGate	100	0	6	77	94.3	100.0	97.1
GPT-4o	100	0	40	43	71.4	100.0	83.3
Gemini 2.5-flash	100	0	73	10	57.8	100.0	73.3
SELP	74	26	12	71	86.0	74.0	79.6
RoboGuard	61	39	21	62	74.4	61.0	67.0

and RoboGuard authorizes 39, allowing dangerous tasks to proceed without intervention.

Our analysis also showed that SafeGate’s deferral behavior was valuable but needs further work. Its $DR-A\% = 28.3$ indicates that it defers on roughly a quarter of ambiguous tasks while still authorizing those it can resolve through structured reasoning, and its $DR-S\% = 4.8$ indicates minimal over-caution on safe tasks. This means that SafeGate’s three-way decision mechanism is selective rather than indiscriminate. The analysis further showed that of the 21 tasks SafeGate defers on, the binary baselines authorize the majority of them. SELP authorizes 16 (76.2%) and RoboGuard authorizes 15 (71.4%). This means that when SafeGate identifies a command as requiring human clarification, for example, “bring the bottle from the fridge to the person in the bedroom” where the type bottle and its content are unknown, SELP and RoboGuard would execute the command without asking for clarification.

(Table I) also show three methods achieving perfect recall (SafeGate, GPT-4o, Gemini), though they differ in how they achieve it. SafeGate produces only 0.06 false positives per true positive, meaning it incorrectly blocks fewer than 1 safe task for every 16 unsafe tasks it correctly catches. GPT-4o produces 0.40 false positives per true positive (6.7× worse), and Gemini produces 0.73 (12.2× worse). This means that the LLM baselines achieve their safety guarantee by casting an excessively wide net by blocking almost everything, including tasks that are clearly safe. They adopt this approach because they lack the structured hazard decomposition process needed to distinguish genuine threats from benign commands.

McNemar’s paired tests confirm that SafeGate’s advantages are statistically significant. In terms of feasibility ($AR-S\%$), SafeGate significantly outperforms GPT-4o ($\chi^2 = 32.1$, $p < 10^{-8}$), Gemini ($\chi^2 = 67.0$, $p < 10^{-15}$), and RoboGuard ($\chi^2 = 9.0$, $p = 0.003$). The SELP comparison does not reach significance ($\chi^2 = 2.6$, $p = 0.109$) because SELP’s feasibility is 7.3% below SafeGate’s.

B. AI2-THOR Simulation Results

Findings from our analysis show that SafeGate achieved $CR\% = 0$ across all three simulation categories, correctly blocking all 30 hazardous tasks without authorizing a single unsafe execution. This means that SafeGate’s neurosymbolic pipeline recognized compositional hazards, incomplete information gaps, and dangerous consequential states equally well. RoboGuard authorized 93.3% of hazardous tasks (28 of 30), with $CR\% = 100$ on both Incomplete Information and Consequential State categories, meaning its pre-authored rules failed entirely when hazards arose from missing information or downstream state changes rather than explicit rule violations. SELP showed a split pattern as it correctly rejected most Consequential State tasks ($CR\% = 10$) and half of Compositional Hazards ($CR\% = 40$), but authorized all 10 Incomplete Information tasks ($CR\% = 100$), revealing that because SELP extracts constraints from the command text alone, it cannot recognize when critical information is absent. Hence, a command like “bring the bottle to the person” contains no constraint violation, yet the hazard lies precisely in what the command does not say. Only SafeGate’s three-way decision mechanism, where the LLM identifies unknowns and the deterministic gate enforces deferral, achieves zero unsafe authorizations across all three reasoning categories.

C. Real-Robot Experiments

Qualitative results from our real-robot experiments showed that our Hazard Library Template was effective in detecting and preventing unsafe task commands. The result also showed that our system prompted the user to create a new hazard template for unmapped hazardous tasks, highlighting the effectiveness of the adaptive component of the system. Furthermore, we evaluated our Task Safety Contract generation, planning verification, and runtime monitoring and found that SafeGate system’s invariant provided a useful runtime shield for LLM-based system.

VII. CONCLUSION

In this research, we introduced *SafeGate*, a neurosymbolic pre-execution safety architecture for LLM-controlled robot systems. Our experimental results showed that SafeGate significantly outperforms baseline approaches, leading to significant reduction in harmful task acceptance while still maintaining high acceptance rate of safe tasks. Future work will explore repairing defective and unsafe task commands instead of rejecting them.

REFERENCES

- [1] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, “Progprompt: Generating situated robot task plans using large language models,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 11 523–11 530.
- [2] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, “Do as i can, not as i say: Grounding language in robotic affordances,” *arXiv preprint arXiv:2204.01691*, 2022.
- [3] S. S. Kannan, V. L. Venkatesh, and B.-C. Min, “Smart-llm: Smart multi-agent robot task planning using large language models,” in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 12 140–12 147.
- [4] R. Wang, D. Zhao, Z. Yuan, I. Obi, and B.-C. Min, “Prefclm: Enhancing preference-based reinforcement learning with crowdsourced large language models,” *IEEE Robotics and Automation Letters*, vol. 10, no. 3, pp. 2486–2493, 2025.
- [5] I. Obi, R. Wang, W. Jo, and B.-C. Min, “Investigating the Impact of Trust in Multi-human Multi-Robot Task Allocation,” *arXiv preprint arXiv:2409.16009*, 2024.
- [6] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar *et al.*, “Inner monologue: Embodied reasoning through planning with language models,” *arXiv preprint arXiv:2207.05608*, 2022.
- [7] S. H. Vemprala, R. Bonatti, A. Buckner, and A. Kapoor, “Chatgpt for robotics: Design principles and model abilities,” *Ieee Access*, vol. 12, pp. 55 682–55 696, 2024.
- [8] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choro-manski, T. Ding, D. Driess, A. Dubey, C. Finn *et al.*, “Rt-2: Vision-language-action models transfer web knowledge to robotic control,” *arXiv preprint arXiv:2307.15818*, 2023.
- [9] Z. Yang, S. S. Raman, A. Shah, and S. Tellex, “Plug in the safety chip: Enforcing constraints for llm-driven robot agents,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14 435–14 442.
- [10] R. Wang, D. Zhao, Z. Yuan, T. Shao, G. Chen, D. Kao, S. Hong, and B.-C. Min, “Primt: Preference-based reinforcement learning with multimodal feedback and trajectory synthesis from foundation models,” in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.
- [11] Y. Wu, Z. Xiong, Y. Hu, S. S. Iyengar, N. Jiang, A. Bera, L. Tan, and S. Jagannathan, “Selp: Generating Safe and Efficient Task Plans for Robot Agents with Large Language Models,” *arXiv preprint arXiv:2409.19471*, 2024.
- [12] *Robots and robotic devices—Safety requirements for personal care robots*, International Organization for Standardization Std. ISO 13 482:2014, 2014. [Online]. Available: <https://www.iso.org/standard/53820.html>
- [13] *Safety of machinery—General principles for design—Risk assessment and risk reduction*, International Organization for Standardization Std. ISO 12 100:2010, 2010. [Online]. Available: <https://www.iso.org/standard/51528.html>
- [14] Z. Ravichandran, A. Robey, V. Kumar, G. J. Pappas, and H. Hassani, “Safety guardrails for llm-enabled robots,” in *RSS 2025 Workshop on Reliable Robotics: Safety and Security in the Face of Generative AI*.
- [15] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, M. Deitke, K. Ehsani, D. Gordon, Y. Zhu *et al.*, “Ai2-thor: An interactive 3d environment for visual ai,” *arXiv preprint arXiv:1712.05474*, 2017.
- [16] X. Wu, R. Xian, T. Guan, J. Liang, S. Chakraborty, F. Liu, B. M. Sadler, D. Manocha, and A. Bedi, “On the safety concerns of deploying llms/vlms in robotics: Highlighting the risks and vulnerabilities,” in *First Vision and Language for Autonomous Driving and Robotics Workshop*, 2024.
- [17] R. Azeem, A. Hundt, M. Mansouri, and M. Brandão, “Llm-driven robots risk enacting discrimination, violence, and unlawful actions,” *arXiv preprint arXiv:2406.08824*, 2024.
- [18] A. Hundt, W. Agnew, V. Zeng, S. Kacianka, and M. Gombolay, “Robots enact malignant stereotypes,” in *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, 2022, pp. 743–756.
- [19] A. Althobaiti, A. Ayala, J. Gao, A. Almutairi, M. Deghat, I. Raz-zak, and F. Cruz, “How can llms and knowledge graphs contribute to robot safety? a few-shot learning approach,” *arXiv preprint arXiv:2412.11387*, 2024.
- [20] S. Izquierdo-Badiola, G. Canal, C. Rizzo, and G. Alenyà, “Plan-col-labnl: Leveraging large language models for adaptive plan generation in human-robot collaboration,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 17 344–17 350.
- [21] I. Obi, R. Pant, S. S. Agrawal, M. Ghazanfar, and A. Basiletti, “Value Imprint: A Technique for Auditing the Human Values Embedded in RLHF Datasets,” in *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- [22] I. Obi, V. L. Venkatesh, W. Wang, R. Wang, D. Suh, T. I. Aмосa, W. Jo, and B.-C. Min, “Safeplan: Leveraging formal logic and chain-of-thought reasoning for enhanced safety in llm-based robotic task planning,” *arXiv preprint arXiv:2503.06892*, 2025.