LabGuard: Grounding Natural-Language Laboratory Rules into Runtime Guards for Embodied Laboratory Agents

Anonymous ACL submission

Abstract

Scientific embodied agents are increasingly capable of carrying out laboratory procedures, but executing these procedures safely in dynamic laboratory environments remains challenging. Current safety approaches often overlook the intermediate step of transforming laboratory natural language, including safety rules, manuals, protocols, and standard operating procedures, into machine-checkable runtime constraints. We introduce **LabGuard**, Laboratory Guard, a language-to-execution safety suite that grounds natural-language laboratory rules into executable specifications and deploys them as runtime guards. LabGuard includes three core components: **LabGuard-IR**, which defines a typed executable representation; **LabGuard-Bench**, which provides 812 supervised annotations expanded from 203 seed laboratory rules; and **LabGuard-Grounder**, which maps natural-language laboratory rules into LabGuard-IR. The resulting IR instances are handled by the **LabGuard Pipeline**, which compiles them into runtime monitors and applies them at the controller boundary. Experiments show that LabGuard generalizes to unseen laboratory-rule sources, achieves 79.4 task-scope F1, and reduces unsafe events from 39.5% to 23.8% after monitor compilation. In LabUtopia, its runtime monitors integrate with ACT, keeping interventions below 0.5% while preserving task success.

1 Introduction

With the rapid development of foundation models, Vision-Language-Action (VLA) models, imitation-learning policies, and related techniques, embodied agents are moving beyond simple manipulation skills toward the execution of complex laboratory procedures. Specifically, VLA models (Brohan et al., 2023; Kim et al., 2024; Physical Intelligence, 2024) enhance robots' ability to generate actions from multimodal inputs; imitation-learning policies, such as ACT (Zhao et al., 2023) and Diffusion Policy (Chi et al., 2025), improve the learning and execution of laboratory manipulation skills; and laboratory simulation platforms and robotic chemistry systems, such as LabUtopia (Li et al., 2025), RoboChemist (Zhang et al., 2025b), and Organa (Darvish et al., 2024), further establish scientific laboratories as an important evaluation setting for embodied agents. However, in laboratory environments, task success does not necessarily imply safe execution, since agents must also follow safety knowledge expressed in natural-language rules, manuals, protocols, and standard operating procedures, covering chemical compatibility, equipment usage, procedural ordering, and risk intervention.

Existing safety research for laboratory embodied agents falls into three lines. First, safety benchmarks test hazard recognition, risk assessment, unsafe-instruction refusal, and safety-aware planning, including LabSafety Bench, LABSHIELD, SafeAgentBench, VESTABENCH, and Safe-BeAl (Zhou et al., 2024; Sun et al., 2026; Yin et al., 2024; Sadhu et al., 2025; Huang et al., 2025). Second, decision-level methods improve planners or policies through safety-aware reasoning, alignment, and constrained learning, such as planning agents, risk-cognition modules, preference alignment, and constrained VLA optimization (Ni et al., 2025; Zhang et al., 2026; Khan et al., 2025a; Zhang et al., 2026). Third, runtime enforcement methods use guardrails, executable predicates, code-based monitors, or control constraints to block, replan, or restrict unsafe actions during execution (Wang et al., 2025b,a; Zhou et al., 2025a; Ames et al., 2019a). However, these works often assume safety information is already formalized. In laboratories, safety knowledge often appears as natural language, including SOPs, manuals, chemical-handling rules, and protocol instructions. How to ground such text into executable monitor specifications for controller-side checking remains under-

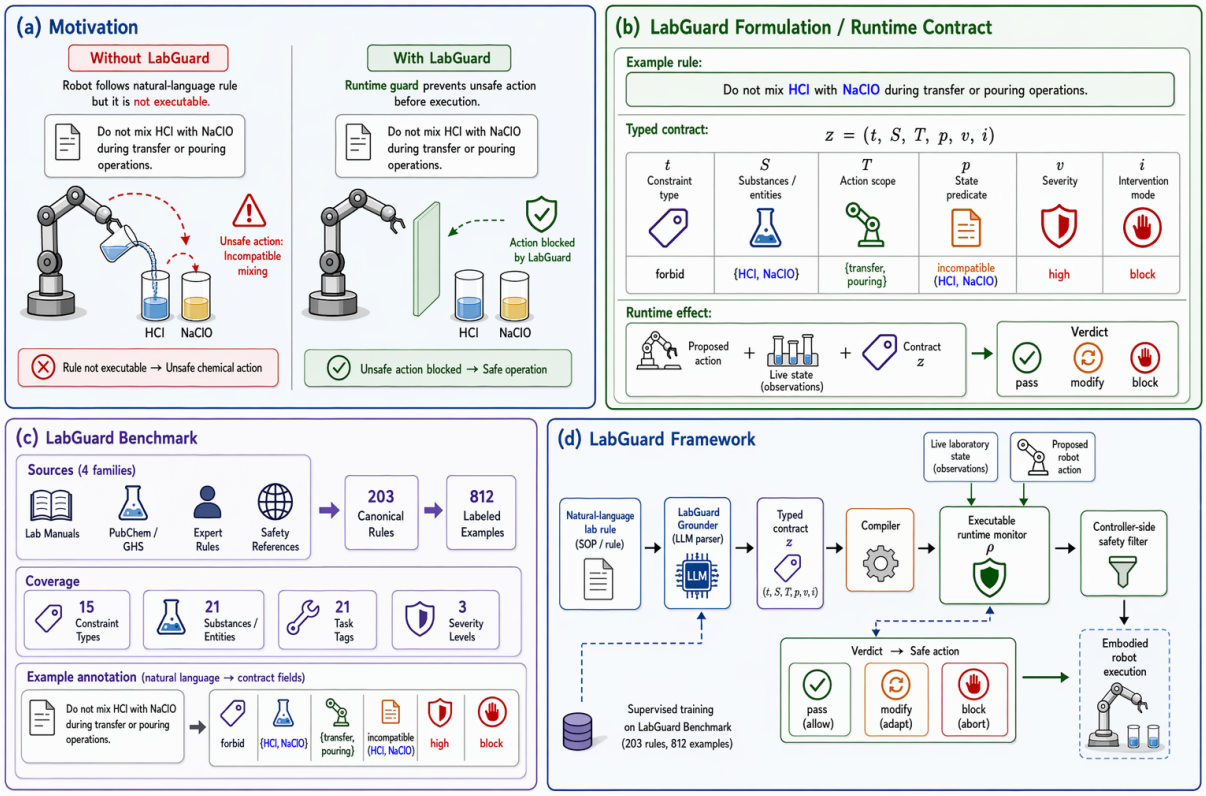

Figure 1: Overview of LabGuard. Natural-language laboratory rules are grounded into LabGuard-IR, compiled into executable runtime monitors, and applied at the controller boundary during embodied laboratory execution.

characterized.

The challenge is not simply to convert text into structured fields, but to make human-readable laboratory safety knowledge enter the robot control loop. This requires a representable, learnable, and executable path from laboratory natural language to controller-side intervention, which raises three key questions. First, how should laboratory safety language be represented so that rules can be activated and checked by a controller? Second, how can models ground diverse sources, including SOPs, manuals, protocols, and chemical safety statements, into a typed executable representation? Third, how can the grounded representation be compiled into runtime monitors that check live laboratory state before robot actions and pass, modify, or block actions when necessary?

To address this problem, we propose **LabGuard**, a language-to-execution safety suite that transforms laboratory natural language into executable runtime guards for embodied laboratory agents. LabGuard includes three core components for executable representation, supervised annotation, and laboratory-language grounding. First, **LabGuard-IR** defines a typed executable representation, specifying the fields needed for runtime monitoring, including constraint type, entity binding, action scope, state predicates, severity, and intervention mode. Second, **LabGuard-Bench** provides 812 supervised annotations for training and evaluation, enabling systematic assessment of how well models recover executable fields from laboratory natural language. Third, **LabGuard-Grounder** maps natural-language laboratory rules into LabGuard-IR, turning human-readable safety knowledge into structured specifications that can be compiled. The resulting IR instances are then handled by the **LabGuard Pipeline**, which compiles them into executable runtime monitors and executes these monitors at the controller boundary, where proposed actions can be passed, modified, or blocked according to the live laboratory state. Experiments show that LabGuard generalizes to unseen laboratory-rule sources with 79.4 task-scope F1 and reduces unsafe events from 39.5% to 23.8% after monitor compilation. In LabUtopia, its ACT-integrated monitors keep interventions below 0.5% while preserving task success.

Our contributions follow the representable, learnable, and executable path introduced above. First,

we propose **LabGuard-IR**, a typed representation that makes laboratory safety language controller-usable by capturing constraint type, entity binding, action scope, state predicates, severity, and intervention mode. Second, we construct **LabGuard-Bench**, which provides 812 supervised annotations expanded from 203 seed laboratory rules, and introduce **LabGuard-Grounder**, which maps natural-language laboratory rules into LabGuard-IR. Third, we develop and evaluate the **LabGuard Pipeline**, which compiles grounded IR instances into runtime monitors and integrates them with LabUtopia + ACT control to test hazard reduction, task preservation, and intervention overhead.

IR field	Grounded value
Natural-language laboratory rule: "Do not mix hydrochloric acid with sodium hypochlorite during transfer or pouring operations."	
$\hat{t}$: constraint type	`chemical_incompatibility`
$\hat{S}$: substances/entities	{*HCl*, *NaClO*}
$\hat{T}$: action scope	{*pour*, *transfer*, *mix*}
$\hat{p}$: state predicate	Destination container must not already contain an incompatible substance.
$\hat{v}$: severity	`high`
$\hat{i}$: intervention	`block`
Runtime effect: If the controller proposes a scoped transfer that would mix HCl and NaClO, the monitor blocks the action before execution.	

Table 1: Example LabGuard-IR grounding. Colored spans in the laboratory rule correspond to the executable fields used to compile a runtime monitor.

2 Related Work

LLM-based Embodied Agent Safety. Recent work on LLM-based embodied agent safety studies how agents recognize unsafe goals, refuse hazardous instructions, assess risk, or generate safety-aware plans (Yin et al., 2024; Ying et al., 2025; Zhang et al., 2025a). These studies typically evaluate safety at the instruction or planning level, where the model is asked to judge whether a goal, instruction, or proposed plan is safe before execution. Related methods further improve decision-level safety through safety-aware reasoning, alignment, constrained learning, or risk-aware planning modules (Khan et al., 2025b; Zhang et al., 2025a).

Runtime Assurance and Safety Filters for Robotics. Runtime assurance and safety-filtering methods aim to restrict unsafe behavior during execution by monitoring states, filtering actions, or enforcing formal constraints at the controller boundary. Classical approaches include control barrier functions and related safety filters that constrain robot actions with respect to predefined safe sets (Ames et al., 2019a,b; Morton and Pavone, 2025). Recent embodied-agent systems also use executable predicates, code-based monitors, visual monitors, or guardrail modules to inspect plans and actions before execution (Zhou et al., 2025b; Wang et al., 2025b,a).

Natural Language Interfaces to Executable Systems. Semantic parsing maps natural language to executable formal representations such as SQL queries, robot commands, logical forms, and API calls (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Tellex et al., 2011; Matuszek et al., 2013). In scientific domains, prior work has extracted structured actions, materials, and experimental operations from synthesis procedures or laboratory protocols (Mysore et al., 2019; Vaucher et al., 2020). Recent structured generation methods further improve output validity through lexical constraints, neural logic decoding, grammar-based parsing, and schema-constrained generation (Hokamp and Liu, 2017; Lu et al., 2022; Scholak et al., 2021; Shin et al., 2021).

3 LabGuard

3.1 Problem Formulation

Laboratory safety knowledge is often written in natural language for human practitioners, including SOPs, safety manuals, chemical handling rules, and protocol instructions, whereas runtime robot controllers require machine-checkable monitor specifications that can be activated, evaluated, and enforced during execution. As illustrated in Figure 1, this requires converting textual safety rules, such as chemical-incompatibility instructions, into typed executable specifications that can be checked against the current laboratory state and controller-proposed actions. We formalize this as a laboratory-language-to-monitor transformation problem:

$$G_\theta(r) = z, \quad E(\text{compile}(z), s_t, a_t) = v_t. \quad (1)$$

Here, r denotes a natural-language laboratory rule, G_θ denotes the grounding model, z denotes the predicted executable safety specification, s_t denotes the live laboratory state, a_t denotes the controller-proposed action, and $v_t \in \{\text{pass}, \text{modify}, \text{block}\}$

denotes the runtime safety verdict. Thus, the goal is not to generate a textual explanation of a safety rule, but to recover the structured execution semantics needed by a runtime monitor.

3.2 LabGuard-IR

We define LabGuard-IR as the typed executable representation used as the target of laboratory-language grounding and the input to runtime monitor compilation. It is designed to capture the execution semantics needed for controller-side safety checking, rather than only describing a rule in natural language. Given a natural-language laboratory rule $r \in \mathcal{R}$, LabGuard-IR represents the grounded executable specification as

$$z = (\hat{t}, \hat{S}, \hat{p}, \hat{T}, \hat{v}, \hat{i}). \tag{2}$$

Here, $\hat{t}$ denotes the constraint type, $\hat{S}$ denotes the substance or entity set, $\hat{p}$ denotes the state or action predicate, $\hat{T}$ denotes the action scope, $\hat{v}$ denotes the severity level, and $\hat{i}$ denotes the intervention mode. Table 1 provides a field-level example of this representation, corresponding to the typed executable view in Figure 1(b).

These fields jointly make the grounded rule executable:

- $\hat{t}$ selects the safety logic to apply, such as incompatibility, temperature, equipment, or sequence checking.
- $\hat{S}$ links textual mentions to concrete laboratory entities that can be tracked in the environment.
- $\hat{p}$ defines the condition evaluated against the live state or proposed action.
- $\hat{T}$ activates the rule only for relevant controller actions.
- $\hat{v}, \hat{i}$ determine the response once a violation is detected, such as logging, modification, or blocking.

3.3 LabGuard-Bench

LabGuard-Bench is a supervised benchmark for training and evaluating laboratory-language grounding, where each natural-language safety rule is aligned to an executable schema: constraint type, canonical substances, formal predicate, robot-task tags, severity, and intervention mode, so each annotation can be compiled into a runtime monitor. The corpus contains **203 seed canonical rules** from four sources: laboratory safety manuals (71), PubChem GHS statements (62), expert-authored controller-facing rules (50), and chemical safety references (20), stratified by difficulty (easy 71, medium 105, hard 27). Seeds are expanded via paraphrase augmentation into **812 labeled examples**. The 15 constraint types group into four families: material compatibility, physical/quantitative handling, process/equipment checks, and environment/PPE/waste. Every seed rule and paraphrase was independently reviewed by all five annotators with simple majority adjudication ($\geq$3 of 5). Agreement distribution and review protocol are in Appendix D.

3.4 LabGuard-Grounder

LabGuard-Grounder is the trainable component that maps natural-language laboratory rules into the LabGuard-IR defined in Eq. 2. Given a supervised example (r_i, z_i) from LabGuard-Bench, where r_i is a natural-language laboratory rule and z_i is the annotated executable specification, the grounder predicts $\hat{z}_i = G_\theta(r_i)$. The predicted IR instance $\hat{z}_i$ follows the same schema as LabGuard-IR and is later passed to the compiler if it satisfies schema validation.

In our implementation, each IR instance z_i is serialized as a structured text sequence, and the grounder is trained with supervised sequence generation. The objective is the negative log-likelihood of the gold IR sequence:

$$\mathcal{L}_{\text{ground}} = -\sum_{i=1}^{N} \sum_{m=1}^{|z_i|} \log P_\theta(z_{i,m} \mid r_i, z_{i,<m}). \tag{3}$$

This objective trains the model to recover the executable fields required for downstream monitor compilation, including constraint type, substance or entity binding, state predicate, action scope, severity, and intervention mode.

At inference time, $\hat{z}_i$ is validated against the LabGuard-IR schema for required fields, categorical values, registry-resolved substances, and compiler-acceptable predicates. Valid IR instances are passed to the compiler, while invalid outputs are rejected or flagged. We implement the grounder with a LoRA fine-tuned language model and evaluate a hybrid variant with rule-based normalization for schema-sensitive fields such as substance mentions. Detailed training hyperparameters are provided in Appendix H.

4 LabGuard Pipeline

Given a LabGuard-IR instance predicted by LabGuard-Grounder, the LabGuard Pipeline turns

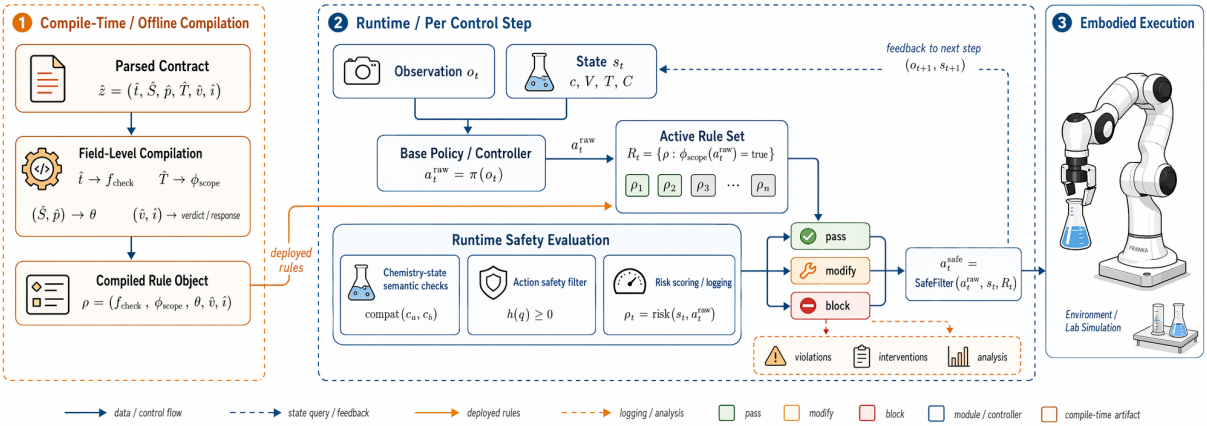

Figure 2: **LabGuard Pipeline.** Given a LabGuard-IR instance predicted by LabGuard-Grounder, the pipeline first compiles it into an executable runtime monitor and then executes the monitor at the controller boundary.

the structured representation into runtime safety behavior. The pipeline consists of two stages: monitor compilation and runtime monitor execution.

4.1 Monitor Compilation

We instantiate the $\text{compile}(\cdot)$ step introduced in Section 3.1. Given a grounded LabGuard-IR instance $\hat{z}$ predicted by LabGuard-Grounder, the compiler lowers it into an executable runtime monitor:

$$\rho = \text{compile}(\hat{z}) = (f_{\text{check}}, \phi_{\text{scope}}, \theta, \hat{v}, \hat{i}). \quad (4)$$

Here, f_{check} is a state-action check function, ϕ_{scope} determines when the monitor is active, θ stores monitor parameters, $\hat{v}$ is the severity level, and $\hat{i}$ is the intervention mode.

The compiler constructs these monitor components from LabGuard-IR fields. The constraint type selects f_{check} from a typed monitor library, e.g., mapping chemical_incompatibility to MaterialCompatibilityCheck. The action scope instantiates ϕ_{scope} so that the monitor is activated only for relevant controller actions. The substance/entity set and predicate populate θ with registry-resolved substances, numeric thresholds, spatial bounds, or other check parameters. The severity and intervention fields determine the logging priority and the runtime response once a violation is detected.

Before producing a monitor, the compiler validates schema conformance, resolves registry entries, and checks whether the IR instance can be lowered to an implemented monitor. IR instances that fail these checks are rejected rather than executed. All accepted monitors expose the same interface, allowing heterogeneous constraint types to be activated and evaluated by the same runtime execution loop.

4.2 Runtime Contract Execution

LabGuard executes compiled monitors at the controller boundary and is agnostic to the underlying policy. At each control step, a base controller π proposes a raw action $a_t^{\text{raw}} = \pi(o_t)$, and the runtime executor uses the live laboratory state s_t to check whether this action satisfies the compiled monitors. Here, s_t denotes the live laboratory state, including per-container substance, volume, temperature, capacity, and robot kinematic state.

The executor first activates the monitors whose action scopes match the proposed action:

$$\mathcal{R}_t = \{\rho \mid \phi_{\text{scope}}^{\rho}(a_t^{\text{raw}}) = \text{true}\}. \quad (5)$$

Each active monitor then evaluates its check function against (s_t, a_t^{raw}) and produces a verdict according to its intervention mode. Given the raw action, live state, and active monitor set, the safety filter returns the executable action:

$$a_t^{\text{safe}} = \text{SafeFilter}(a_t^{\text{raw}}, s_t, \mathcal{R}_t). \quad (6)$$

The safety filter may pass the raw action unchanged, modify it, or block it before execution; all interventions and violations are logged for analysis.

In our implementation, SafeFilter combines chemistry-state-aware semantic checks, a barrier-style action filter, and risk scoring. The semantic checks evaluate IR predicates over live container states, including incompatibility, overflow, temperature, sequencing, PPE, and waste handling; the action filter enforces workspace, joint-margin, and

Method	Tag F1	Pred Soft	Pred EM	Unsafe Event ↓	False Interv. ↓	Task Success
Source-held-out laboratory-language grounding (avg. 4 folds, mean±std):						
Regex BL	58.7±9.2	28.5±12.3	15.8±9.7	34.2±8.5	8.3±3.2	72.4±7.8
SciBERT	54.9±10.8	14.2±7.1	7.6±4.8	41.7±9.3	12.5±4.7	68.3±8.9
Qwen-7B 0-shot	61.3±11.5	20.4±9.8	10.2±6.5	38.9±8.7	10.7±4.1	70.8±8.2
Qwen3-32B 0-shot	64.8±10.9	24.7±10.5	13.1±7.2	35.6±8.2	9.8±3.8	73.2±7.5
DeepSeek-V3.1 0-shot	67.2±10.3	27.3±11.1	14.9±7.8	33.8±7.9	9.2±3.5	74.6±7.1
Qwen2.5-72B 0-shot	68.9±9.8	29.1±11.6	16.3±8.3	32.4±7.6	8.7±3.3	75.8±6.8
Base Grounder (Qwen3-8B)	62.4±11.2	18.7±8.9	11.4±6.8	39.5±8.9	11.3±4.3	69.7±8.5
+ LoRA fine-tuning	75.3±8.7	31.8±10.2	19.6±7.9	28.7±7.2	7.5±2.9	78.4±6.3
+ Constraint decoding	77.8±8.2	34.2±9.8	21.3±7.5	26.3±6.8	6.9±2.6	79.8±5.9
LabGuard-Hybrid	**79.4±7.9**	**36.7±9.5**	**23.1±7.2**	**23.8±6.4**	**6.2±2.4**	**81.2±5.6**
Chemistry-held-out laboratory-language grounding (hydrochloric acid, N=9, 5 seeds):						
Regex BL	52.8±14.3	35.2±16.8	18.5±12.7	42.7±15.2	9.8±5.3	65.3±13.8
Qwen2.5-72B 0-shot	73.6±11.9	40.3±15.4	38.7±14.2	31.5±12.8	8.4±4.7	74.2±11.5
Base Grounder (Qwen3-8B)	68.4±13.5	32.8±16.2	28.3±14.8	36.9±13.7	10.5±5.1	70.8±12.9
+ LoRA fine-tuning	88.7±9.2	68.4±12.8	52.3±15.1	18.6±9.4	5.7±3.8	86.4±8.7
+ Constraint decoding	90.2±8.5	71.5±12.1	54.8±14.6	16.2±8.7	5.1±3.5	87.9±8.2
LabGuard-Hybrid	**91.8±7.9**	**74.3±11.6**	**57.2±14.2**	**14.3±8.1**	**4.6±3.2**	**89.3±7.6**

Table 2: Laboratory-language grounding and compiled-monitor runtime outcomes under held-out settings. Grounding columns (Tag F1, Pred Soft, Pred EM) are evaluated on each held-out test set Appendix A; runtime columns (Unsafe Event, False Interv., Task Success) aggregate LabUtopia L1–L4 episodes after monitor compilation Appendix F.

velocity bounds; and the risk scorer flags high-risk steps from velocity, proximity, chemical-hazard, and uncertainty signals. Full details are provided in Appendix B.

5 Experimental

5.1 Experimental Setup

Benchmarks and evaluation settings. We evaluate LabGuard across three stages, using both our constructed benchmarks and existing runtime environments. For laboratory-language grounding, we use LabGuard-Bench under source-held-out, random 80/20, and chemistry-held-out settings. For compiled-monitor replay, we construct a 100-scenario stress replay set with 70 unsafe scenarios and 30 safe-control scenarios across 7 hazard categories. For runtime monitor execution, we use LabUtopia Levels 1–4 with ACT as the base policy. We additionally use a controlled pour scenario for module ablation. Full benchmark and setting details are provided in Appendix A, with runtime tasks and stress construction described in Appendix F and Appendix G.

Metrics. For laboratory-language grounding, we report Type Accuracy, Substance F1, Task Tag F1, Predicate Soft Match, Predicate Canonical EM, Schema Valid, and Compile Ready. For compiled-monitor replay, we report check-function accuracy, task-scope F1, parameter recovery, catch rate, precision, and F1 on unsafe and safe-control scenarios. For runtime monitor execution, we report Success Rate (SR), Violation Rate (VR), Intervention Rate (IR), Collision Rate (CR), mean Risk, Episode Length (EL), and Task Completion Time (TCT). Detailed metric definitions are provided in Appendix C.

Baselines. For laboratory-language grounding, we compare against a Regex baseline, prompted LLMs, LoRA fine-tuned models, and our hybrid grounding variant. For compiled-monitor replay, we compare monitors compiled from different grounding outputs, including Regex, LoRA, Hybrid, and gold labels / monitors compiled from gold labels. For runtime monitor execution, we compare Base, which uses ACT (Zhao et al., 2023) without the safety layer, against different runtime module configurations. Full implementation details are provided in Appendix H.

5.2 Laboratory-Language Grounding

We first evaluate whether LabGuard can ground natural-language laboratory rules into executable IR instances under distribution shift. Table 2 reports source held-out and chemistry-held-out results, while the random 80/20 split is reported in Appendix E as an in-distribution upper bound.

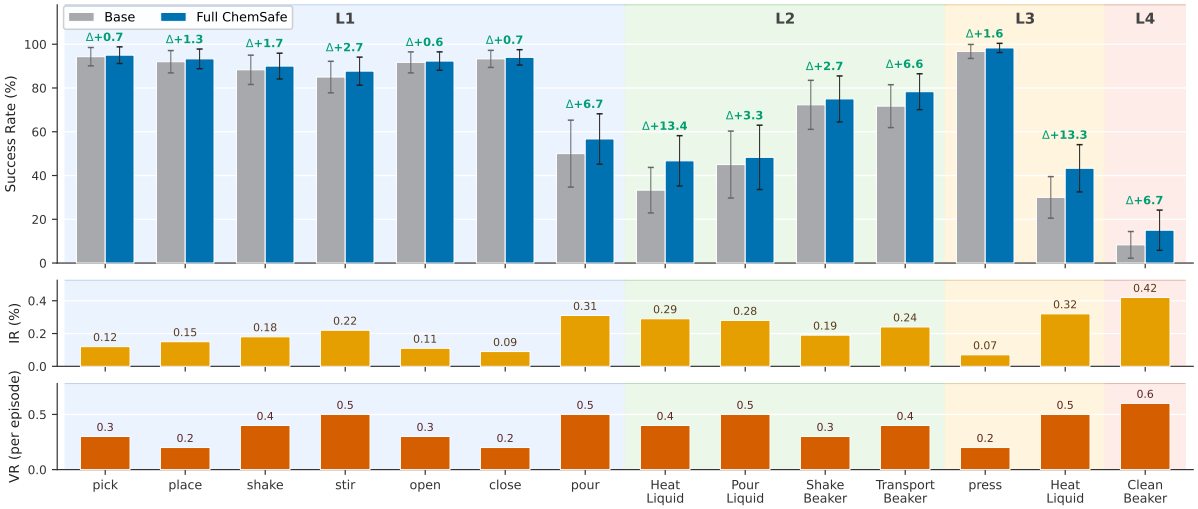

Figure 3: **Per-task success rate (Base vs. Full LabGuard) across L1–L4.** Mean SR over 3 seeds $\times$ 10 episodes (error bars: seed std), with IR and VR under Full and right-hand Δ values reporting Full $-$ Base.

LabGuard-Hybrid achieves the strongest performance, reaching 79.4 task-scope F1 under source-held-out evaluation and 91.8 under chemistry-held-out evaluation. This indicates that LabGuard can still recover when safety rules should be activated even when source families or chemistry categories change.

LabGuard-Hybrid is effective because it combines learned task-scope prediction with rule-based substance normalization. The learned component handles variation in laboratory-rule wording, while the rule-based component reduces substance-resolution errors that can prevent IR instances from compiling correctly. Although exact predicate recovery remains challenging, the grounded IR instances reduce unsafe events from 39.5% to 23.8% after monitor compilation under source-held-out evaluation. This suggests that accurate activation scope and reliable substance binding carry much of the downstream safety signal by routing proposed actions to the correct runtime checks.

5.3 Compiled-Monitor Replay

We next evaluate whether grounded IR instances can be compiled into deterministic monitors that prevent unsafe outcomes in offline replay. The stress set contains 70 unsafe scenarios across 7 hazard categories and 30 safe-control scenarios; among the unsafe cases, 54 are in-schema and executable by the current monitor library, while 16 require additional monitor types such as exothermic-reaction tracking. Gold IR labels catch all 54 in-schema unsafe scenarios, and LoRA and LabGuard-Hybrid approach this oracle bound, with remaining gaps mainly caused by grounding errors rather than monitor-compilation failures Appendix G. The 16 out-of-schema cases are missed by all configurations, including gold, indicating a schema-coverage boundary rather than a translation failure. Table 4 further shows that source-held-out grounding preserves the relative ordering of methods after compilation, although cross-source degradation remains non-trivial.

Config.	TP	FN	FP	Prec	Catch	F1
No Safety	7	63	1	87.5	10.0%	17.9
Regex	34	36	9	79.1	48.6%	60.2
LoRA (Qwen3-8B)	41	29	7	85.4	58.6%	69.5
LabGuard-Hybrid	41	29	7	85.4	58.6%	69.5
Gold	54	16	4	93.1	77.1%	84.4

Table 3: Stress-benchmark results under compiled-monitor replay. Outcome-level catch, precision over the combined unsafe + safe-control pool, and their F1,

We analyze how grounding errors propagate through the LabGuard pipeline and affect downstream execution. Grounding errors mainly concentrate in formal predicate composition, while task-scope grounding remains the most transferable signal under source shifts Table 4. This pattern reflects three factors: schema complexity from 15 heterogeneous constraint types, sparse supervision with only 3–5 paraphrases per canonical rule, and metric limitations where token-level Soft Match may miss semantically equivalent predicates such as "temp $>$ 80" and "80 $<$ temp". Executable equivalence testing is left to future work Appendix I)·.

Grounding Module	Tag F1	Pred Soft	Catch%	Prec	F1
Source-held-out (avg. 4 folds):					
Regex	45.1	33.3	47.1	77.8	58.7
LoRA (Qwen3-8B)	75.7	20.6	56.8	83.4	67.5
LabGuard-Hybrid	76.5	20.4	58.2	84.1	68.7
Random 80/20 (upper bound):					
Regex	68.3	54.2	48.6	79.1	60.2
LoRA (Qwen3-8B)	93.4	87.2	58.6	85.4	69.5
LabGuard-Hybrid	93.8	87.5	58.6	85.4	69.5
Gold	100.0	100.0	77.1	93.1	84.4

Table 4: **Source-held-out propagation to stress benchmark.** Same 70/30 pool as Table 3, re-grounded with source-held-out LoRA.

Configuration	SR↑	IR	VR↓	Risk	CR↓	EL	TCT
Base (no safety)	10%	–	2.8	0.04	1.9	487	24.1s
Planning only	20%	0.8%	2.1	0.05	1.6	512	25.3s
Semantic only	10%	1.2%	1.5	0.05	1.3	495	24.8s
Action Filter only	30%	2.1%	0.9	0.07	0.7	531	26.5s
Risk Scoring only	10%	0.3%	2.5	0.12	2.1	476	23.7s
Full LabGuard	40%	3.2%	0.6	0.08	0.5	548	27.4s

Table 5: **Module-level ablation on the controlled *pour* stress scenario.** SR, IR, VR, Risk, CR, EL, TCT for each module in isolation and the full system.

The compiled pipeline remains effective because runtime safety depends heavily on correct task-scope routing. Once the system identifies when a rule should fire, deterministic check functions and registry-resolved substances recover much of the remaining execution semantics. LabGuard-Hybrid further reduces substance-extraction failures by using regex fallback for the substance slot. This explains why models with modest predicate scores can still yield measurable safety improvements after compilation.

5.4 Runtime Monitor Execution

The final stage closes the loop: compiled monitors must attach to learned controllers in a live simulator without degrading task success. We evaluate LabGuard on LabUtopia Levels 1–4 (per-task list in Appendix F; full numbers in Appendix F) to demonstrate runtime feasibility and bound the overhead of compiled-monitor execution. The focus is on whether the safety layer can integrate with ACT policies without task-success loss; main-runtime VR/Unsafe Event are assurance-layer-logged (Appendix C), while an outcome-level oracle independent of the wrapper is applied only to the controlled pour ablation (Table 5).

Figure 3 shows the safety layer is essentially a no-op on tasks already at ceiling and is associated with the largest task-success gains on safety-relevant operations. Intervention rates stay $\leq 0.5\%$ across tasks, and assurance-layer-logged violations remain in the 0.2–0.6 range. This pattern confirms that compiled monitors do not introduce spurious interventions or degrade controller performance, completing the path from laboratory language to runtime monitoring without controller-side friction.

5.5 Runtime Module Ablation

We isolate the contribution of each runtime module using a controlled *pour* stress scenario that deliberately triggers unsafe behavior under a weak base policy. This offline-rescored ablation uses an outcome-level oracle independent of the runtime assurance layer, and Table 5 reports success rate (SR), intervention rate (IR), violation rate (VR), risk score, collision rate (CR), episode length (EL), and total cycle time (TCT). Full LabGuard improves SR from 10% to 40% and reduces logged violations from 2.8 to 0.6 per episode. Among individual modules, the Action Filter contributes the largest gain, raising SR to 30% and reducing VR to 0.9, while Planning and Semantic modules provide complementary violation reductions. Risk Scoring alone does not improve SR, but provides discriminative risk information for future adaptive interventions. Overall, the ablation shows that runtime safety comes from the combination of task-scope routing, chemistry-state-aware checks, and kinematic filtering rather than any single module.

6 Conclusion

We presented LabGuard, a language-to-execution safety suite that transforms laboratory natural language into typed executable specifications and runtime guards for controller-side intervention in simulation. By separating laboratory-language grounding, monitor compilation, and runtime execution, LabGuard provides an auditable path from human-readable laboratory safety knowledge to executable monitors. Through LabGuard-Bench and a three-stage evaluation protocol, we show that laboratory-language grounding and monitor compilation can link language-level safety rules to runtime robot behavior within a schema-scoped setting. Future work should expand the monitor schema, improve cross-source predicate grounding, and evaluate stronger safety guarantees beyond simulation.

Limitations

Our work is limited by its simulation-only evaluation. All runtime experiments and stress tests are conducted in the LabUtopia virtual environment, so the results show that LabGuard can connect laboratory natural language to runtime monitors in simulation, but they do not establish real-world laboratory safety. Physical deployment would require additional validation with real robot dynamics, sensing noise, calibration errors, and institutional safety procedures. A second limitation is broader policy coverage. We use ACT as the main learned controller in LabUtopia, which is sufficient to test whether compiled monitors can be inserted at the controller boundary. Future work can evaluate the same runtime-contract layer with more VLA models and manipulation policies to study how the safety layer behaves across different embodied agents.

Ethics Statement

This work is conducted entirely in simulation and does not involve deployment on physical laboratory hardware or interaction with real hazardous chemicals. As a result, the experiments do not create direct real-world safety risks during evaluation. At the same time, the paper studies safeguards for scientific robotics, so we intentionally avoid overstating claims to reduce the risk of misleading conclusions about real-world laboratory safety.

References

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

Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. 2019b. Control barrier functions: Theory and applications. *European Control Conference (ECC)*.

Anthony Brohan, Noah Brown, Justice Carbajal, et al. 2023. RT-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. 2025. Diffusion policy: Visuomotor policy learning via action diffusion. *International Journal of Robotics Research*, 44(10-11).

Kourosh Darvish, Marta Skreta, Yuchi Zhao, Naruki Yoshikawa, Sagnik Som, Miroslav Bogdanovic, Yang Cao, Han Hao, Haoping Xu, Alán Aspuru-Guzik, Animesh Garg, and Florian Shkurti. 2024. ORGANA: A robotic assistant for automated chemistry experimentation and characterization. *Matter*.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *ACL*.

Yuting Huang, Leilei Ding, Zhipeng Tang, Tianfu Wang, Xinrui Lin, Wuyang Zhang, Mingxiao Ma, and Yanyong Zhang. 2025. A framework for benchmarking and aligning task-planning safety in llm-based embodied agents. *arXiv preprint arXiv:2504.14650*.

Azal Ahmad Khan, Michael Andrev, Muhammad Ali Murtaza, Sergio Aguilera, Rui Zhang, Jie Ding, Seth Hutchinson, and Ali Anwar. 2025a. Safety aware task planning via large language models in robotics. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 21024–21031. IEEE.

Azal Ahmad Khan et al. 2025b. Safety aware task planning via large language models in robotics. *arXiv preprint arXiv:2503.15707*.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, et al. 2024. OpenVLA: An open-source vision-language-action model. In *CoRL 2024*.

Rui Li, Zixuan Hu, Wenxi Qu, Jinouwen Zhang, et al. 2025. LabUtopia: High-fidelity simulation and hierarchical benchmark for scientific embodied agents. In *NeurIPS 2025 Datasets and Benchmarks Track*.

Ximing Lu, Sean Welleck, Peter Liber, Hannaneh Hajishirzi, and Yejin Choi. 2022. NeuroLogic A*esque decoding: Constrained text generation with lookahead heuristics. In *NAACL*.

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. 2013. Learning to parse natural language commands to a robot control system. In *ISER*.

Daniel Morton and Marco Pavone. 2025. Safe, task-consistent manipulation with operational space control barrier functions. In *IROS 2025*.

Sheshera Mysore, Zach Jensen, Edward Kim, Kevin Huang, Haw-Shiuan Chang, Emma Strubell, Jeffrey Flanigan, Andrew McCallum, and Elsa Olivetti. 2019. The materials science procedural text corpus: Annotating materials synthesis procedures with shallow semantics. In *LAW*.

Minheng Ni, Lei Zhang, Zihan Chen, Kaixin Bai, Zhaopeng Chen, Jianwei Zhang, and Wangmeng Zuo. 2025. Don't let your robot be harmful: Responsible robotic manipulation via safety-as-policy. *IEEE Robotics and Automation Letters*.

Physical Intelligence. 2024. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*.

Tanmana Sadhu, Yanan Chen, and Ali Pesaranghader. 2025. Vestabench: An embodied benchmark for safe long-horizon planning under multi-constraint and adversarial settings. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 2122–2145.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *EMNLP*.

Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *EMNLP*.

Qianpu Sun, Xiaowei Chi, Yuhan Rui, Ying Li, Kuangzhi Ge, Jiajun Li, Sirui Han, and Shanghang Zhang. 2026. Labshield: A multimodal benchmark for safety-critical reasoning and planning in scientific laboratories. *arXiv preprint arXiv:2603.11987*.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*.

Alain C. Vaucher, Federico Zipoli, Joppe Geber, and Teodoro Laino. 2020. Automated extraction of chemical synthesis actions from experimental procedures. *Nature Communications*, 11:3601.

Haoyu Wang, Christopher M Poskitt, and Jun Sun. 2025a. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666*.

Le Wang, Zonghao Ying, Xiao Yang, Quanchen Zou, Zhenfei Yin, Tianlin Li, Jian Yang, Yaodong Yang, Aishan Liu, and Xianglong Liu. 2025b. Robosafe: Safeguarding embodied agents via executable safety logic. *arXiv preprint arXiv:2512.21220*.

Sheng Yin et al. 2024. SafeAgentBench: A benchmark for safe task planning of embodied LLM agents. *arXiv preprint arXiv:2412.13178*.

Zonghao Ying, Le Wang, Yisong Xiao, Jiakai Wang, Yuqing Ma, Jinyang Guo, Zhenfei Yin, Mingchuan Zhang, Aishan Liu, and Xianglong Liu. 2025. AGENTSAFE: Benchmarking the safety of embodied agents on hazardous instructions. *arXiv preprint arXiv:2506.14697*.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *AAAI*.

Luke Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*.

Borong Zhang, Yuhao Zhang, Jiaming Ji, Yingshan Lei, Juntao Dai, Yuanpei Chen, and Yaodong Yang. 2026. Safevla: Towards safety alignment of vision-language-action model via constrained learning. *Advances in Neural Information Processing Systems*, 38:153335–153373.

Borong Zhang, Yuhao Zhang, Jiaming Ji, and Yaodong Yang. 2025a. SafeVLA: Towards safety alignment of vision-language-action model via constrained learning. In *NeurIPS 2025 (Spotlight)*.

Zongzheng Zhang, Chenghao Yue, Haobo Xu, Minwen Liao, Xianglin Qi, Huan-ang Gao, Ziwei Wang, and Hao Zhao. 2025b. Robochemist: Long-horizon and safety-compliant robotic chemical experimentation. *arXiv preprint arXiv:2509.08820*.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. 2023. Learning fine-grained bimanual manipulation with low-cost hardware. In *RSS 2023*.

Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, and He Wang. 2025a. Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 6919–6929.

Enshen Zhou, Qi Su, Cheng Chi, Zhizheng Zhang, Zhongyuan Wang, Tiejun Huang, Lu Sheng, and He Wang. 2025b. Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Yujun Zhou, Jingdong Yang, Kehan Guo, Pin-Yu Chen, Tian Gao, Werner Geyer, Nuno Moniz, Nitesh V. Chawla, and Xiangliang Zhang. 2024. LabSafety Bench: Benchmarking LLMs on safety issues in scientific labs. *arXiv preprint arXiv:2410.14182*.

A Evaluation Matrix

Table 6 summarizes the evaluation settings used in the main paper. We separate three kinds of evidence: laboratory-language grounding on held-out labels, offline replay of compiled monitors on scripted safety scenarios, and live LabUtopia execution with ACT. "Offline replay" means that compiled monitors are evaluated on scripted state traces; "live runtime" means that the safety layer runs inside the controller loop. This distinction is important because the stress benchmark uses outcome-level replay, whereas the L1–L4 runtime results report events logged by the active assurance layer.

Protocol	Data & Split	N	Primary Metric(s)	Grounding Ckpt	Mode
Grounding, random split	LabGuard-Bench, seed-42 80/20 split	649 / 163 examples	Type Acc, Subst F1, Tag F1, Pred Soft/EM	per method	held-out labels
Grounding, source-held-out	LabGuard-Bench, leave-one-source-out	4 folds	Tag F1, Pred Soft/EM	per method	held-out labels
Grounding, chemistry-held-out	HCl rules held out from training	9 rules, 5 seeds	Tag F1, Pred Soft/EM	per method	held-out labels
Replay diagnostics	41 held-out rules + scripted scenarios	80 scenarios	Check Acc, Catch, Prec, Safety	per method	offline replay
Stress benchmark	7 hazard categories + safe controls	70 unsafe / 30 safe	TP, FN, FP, Catch, Prec, F1	per method or gold	offline replay
in-schema subset	stress cases covered by implemented monitors	54 unsafe / 30 safe	Catch, Prec, F1	per method or gold	offline replay
out-of-schema subset	stress cases requiring new monitor types	16 unsafe	missed by current schema	per method or gold	offline replay
Stress propagation	same stress pool re-grounded by held-out models	70 unsafe / 30 safe	Tag F1, Pred Soft, Catch, Prec, F1	source-held-out models	offline replay
Full LabGuard	L1–L4 tasks with ACT controller	3 seeds × 10 episodes per task	SR, IR, VR, unsafe-event rate	full safety layer	live runtime
Module ablation	controlled adversarial *pour* scenario	3 seeds × 10 episodes	SR, IR, VR, Risk, CR, EL, TCT	module variants	live runtime + offline VR rescoring

Table 6: Evaluation matrix. Summary of the grounding, offline replay, and live-runtime protocols used in the paper.

B Runtime Safety Module Details

This appendix gives additional implementation details for the runtime monitor execution modules summarized in Section 5.4. The modules operate after laboratory-language grounding and monitor compilation. No language model is invoked inside the controller loop.

B.1 Compiled Monitor Target

Given a grounded LabGuard-IR instance

$$\hat{z} = (\hat{t}, \hat{S}, \hat{p}, \hat{T}, \hat{v}, \hat{i}),$$

the compiler lowers it into a runtime monitor

$$\rho = \mathrm{compile}(\hat{z}) = (f_{\mathrm{check}}, \phi_{\mathrm{scope}}, \theta, \hat{v}, \hat{i}).$$

Here, f_{check} is the state-action check function, ϕ_{scope} determines whether the monitor is active for a proposed action, θ stores monitor parameters, $\hat{v}$ is the severity level, and $\hat{i}$ is the intervention mode. The scope predicate is instantiated from the action-scope field:

$$\phi_{\mathrm{scope}}(a) = \mathbf{1}[\mathrm{tag}(a) \in \hat{T}].$$

The constraint type $\hat{t}$ selects a monitor class from the typed monitor library, while $\hat{S}$ and $\hat{p}$ populate θ with registry-resolved substances, thresholds, spatial bounds, or other check parameters.

For material-compatibility specifications, the monitor uses an incompatibility table over registered substances. For two substances c_a and c_b, the lookup is

$$\mathrm{compat}(c_a, c_b) = \begin{cases} 1, & c_b \notin \mathrm{inc}(c_a) \wedge c_a \notin \mathrm{inc}(c_b), \\ 0, & \text{otherwise}, \end{cases} \tag{7}$$

where $\mathrm{inc}(\cdot)$ denotes the incompatible-substance list in the runtime registry. This lookup is intentionally simple. Violations are triggered by explicit registry entries and task-conditioned monitor templates, not by a symbolic planner or reaction simulator.

B.2 Chemistry-State-Aware Semantic Checks

The runtime assurance layer maintains a live laboratory state containing registered containers, their substance identities, volumes, temperatures, capacities, and the robot kinematic state. Active monitors evaluate their predicates against this state and the controller-proposed action.

For transfer-like actions, the monitor inspects the source container, destination container, transferred substance, destination contents, and planned transfer volume. The supported checks include incompatible mixing, destination overflow, temperature-threshold violations, sequencing preconditions, PPE-related preconditions, and waste-handling constraints. If a violation is detected, the monitor returns a verdict according to its intervention mode. The action may be blocked or modified before execution, and the intervention is logged for analysis. If no violation is detected, the state update is committed so that later monitors operate on the updated container state.

B.3 Geometric and Kinematic Bounds

The runtime layer also applies lightweight geometric and kinematic validation. These checks enforce fixed workspace bounds, conservative joint margins, and per-joint velocity limits. The workspace constraint is applied componentwise:

$$p_{\min,k} \le p_{ee,k} \le p_{\max,k}, \qquad k \in \{x, y, z\}, \tag{8}$$

where p_{ee} is the end-effector position. The action-side velocity constraint is

$$|\dot{q}_j| \le \alpha \dot{q}_{j,\max}, \qquad \alpha = 0.8. \tag{9}$$

These bounds are fixed in the current implementation and are used as conservative runtime safeguards rather than learned safety constraints.

B.4 Barrier-Style Action Filter

The action filter uses barrier-style margins around joint limits together with velocity clamping. For

joint j, the margins are

$$h_j^+(q) = q_{j,\max} - q_j - \delta, \quad (10)$$

$$h_j^-(q) = q_j - q_{j,\min} - \delta, \quad (11)$$

where δ is a fixed safety margin. These quantities are used to steer target positions away from joint boundaries and to clamp velocities. Although the design is inspired by control-barrier constraints, the implementation does not solve a quadratic program and does not claim the guarantees of a full optimization-based CBF controller.

B.5 Risk Scoring and Logging

The risk scorer aggregates velocity, proximity, chemical hazard, and uncertainty with fixed hand-designed weights:

$$\rho_t = 0.25\rho_v + 0.25\rho_p + 0.30\rho_c + 0.20\rho_u. \quad (12)$$

The coefficients are fixed heuristics rather than learned parameters. When chemistry-state-aware checks are enabled, the chemical term ρ_c is computed from the live container state, including substance identities, volumes, fill fractions, and temperature-related signals. High-risk steps are logged for analysis, and threshold exceedances can produce safety events according to the active monitor configuration. In the current implementation, risk scoring is primarily diagnostic: it flags high-risk states and supports intervention analysis, but it does not by itself provide a formal safety guarantee or restore simulator state.

C Runtime Metric Definitions

We report runtime metrics under two scoring modes. For L1–L4 LabUtopia runs, safety events are logged by the runtime assurance layer when that layer is active. For the controlled *pour* ablation, we additionally rescore each trajectory with an offline compiled-monitor checker, so that the Base policy, which has no active safety layer, can be compared with the safety-module variants.

Success Rate. SR is the fraction of episodes that complete the task. Higher SR indicates better task performance.

Violation Rate. VR is the average number of safety violations per episode. In the L1–L4 runtime table, VR counts violations logged by the active assurance layer. In the controlled *pour* ablation, VR is computed by replaying each trajectory through the same compiled-monitor checker. This offline scoring makes the Base row comparable to rows with an active safety layer.

Intervention Rate. IR is the fraction of controller steps in which the safety layer blocks, modifies, or replaces the proposed action. It measures how often the runtime layer changes controller behavior. Lower IR is preferable when safety outcomes are comparable.

Unsafe Events and False Interventions. Unsafe Event is reported for runtime-style summaries as the percentage of episodes in which the scoring protocol observes an unsafe outcome. In offline replay settings, the analogous outcome-level quantities are reported as catch, precision, and F1. False Intervention measures interventions on safe-control cases or safe execution segments, and is reported only when such safe-control labels are available.

Risk, Collision, and Runtime Cost. Risk is the mean heuristic risk score from the runtime scorer. CR is the average number of collision events per episode in the controlled ablation. EL is the episode length in control steps, and TCT is the total cycle time. These quantities are reported in the controlled ablation to describe side effects of each runtime module.

D LabGuard-Bench Construction Protocol

D.1 Annotation Protocol

The corpus starts from 203 seed rules from four source families described in Section 3.3. Each seed is labeled with constraint type, canonical substances, predicate, task tags, severity, and intervention mode. Paraphrases inherit the seed label only after review for semantic equivalence.

Five annotators with chemistry or robotics background reviewed the seed labels and paraphrases. Final labels use simple majority agreement. Items without majority agreement are revised before inclusion. This protocol is intended to keep paraphrase augmentation from adding or removing safety constraints.

We keep source-family metadata for each item and do not rely on verbatim source text in the released labels. Table 7 reports the final distribution over the 15 constraint types. The counts sum to 812 examples. Per-source seed counts are 71 lab-manual rules, 62 PubChem GHS statements,

Constraint type	N	Constraint type	N
chemical_incompatibility	132	velocity_limit	40
equipment_check	100	waste_handling	36
sequence_constraint	80	spill_prevention	36
exposure_limit	68	container_capacity	24
temperature_constraint	64	storage_constraint	24
ventilation_requirement	60	workspace_zone	20
fire_safety	56	force_limit	20
ppe_requirement	52	**Total**	**812**

Table 7: **Per-constraint-type distribution** across the 812-example LabGuard-Bench corpus.

50 expert-authored controller-facing rules, and 20 chemical-safety reference rules.

For source-held-out evaluation, all seed rules and paraphrases from the held-out source family are removed from training. Chemical names and constraint types can still appear across folds because they belong to the shared schema and registry. The split is therefore designed to test source phrasing and rule-structure shift, not unseen chemistry vocabulary alone.

E Grounding Split Details

The random 80/20 split uses seed 42 over the 812-example LabGuard-Bench corpus. It can share source family, vocabulary, and paraphrase neighbors between train and test, so we treat it as an in-distribution upper bound. The source-held-out split is the main cross-source protocol for laboratory-language grounding.

E.1 Source-Held-Out Per-Fold Breakdown

Table 10 reports the four folds used in the source-held-out protocol. The main table reports the corresponding mean and standard deviation.

F Runtime Evaluation Details

The ACT base policy follows the standard ACT setup used in the main runtime experiments. It uses two RGB camera views at 256×256 resolution and robot state, and each task model is trained from roughly 50 scripted demonstrations. This small demonstration budget is one reason the runtime results have high variance on long-horizon or safety-sensitive tasks.

We evaluate on LabUtopia Levels 1–4. Level 1 includes pick, place, press, shake, stir, open, close, and pour. Level 2 includes HeatLiquid, PourLiquid, ShakeBeaker, and TransportBeaker. Level 3 uses OOD versions of press and HeatLiquid. Level 4 uses CleanBeaker. Each task is evaluated with 3 random seeds and 10 episodes per seed. Table 11 gives the per-task results.

G Replay and Stress Benchmark Details

Table 12 reports replay diagnostics on 41 held-out rules and 80 scripted scenarios. These diagnostics use random-split grounding checkpoints, so they should be read as an in-distribution replay setting rather than the source-held-out operating point.

The stress benchmark contains 70 unsafe scenarios and 30 safe-control scenarios. The unsafe cases cover seven categories with 10 scenarios each. Table 13 reports true positives for unsafe categories and false positives for safe controls.

Stress scenarios are constructed to match the categories used in Table 3. Safe controls preserve surface features of unsafe cases while removing the violation. The replay evaluator runs compiled monitors on scripted state traces and records whether the unsafe outcome is caught.

H Implementation Details

The LoRA laboratory-language grounding module uses Qwen3-8B as the base model. We train with rank r=16, alpha α=32, target modules `q_proj`, `k_proj`, `v_proj`, and `o_proj`, 3 epochs, batch size 8, AdamW, learning rate 2×10^{-4}, fp16 precision, and gradient accumulation of 4 steps. Decoding is greedy with temperature 0 and a maximum of 512 new tokens.

The regex baseline uses keyword scoring for the 15 constraint types, a curated chemical-name dictionary for substance extraction, template rules for task tags, and template instantiation for predicates. It has no learned component.

I Future Predicate Evaluation Protocols

Current token-level metrics do not fully capture executable semantic equivalence. Future evaluations should compare predicates after normalization or by execution.

I.1 Canonical AST Exact Match

Predicates can be normalized before comparison.

- Reorder commutative operands: $[\mathrm{HCl}, \mathrm{H_2SO_4}] \equiv [\mathrm{H_2SO_4}, \mathrm{HCl}]$
- Normalize comparison direction: $\textit{temp} > 80 \equiv 80 < \textit{temp}$
- Standardize variable names: $\mathrm{temperature(beaker)} \equiv \mathrm{temp(container)}$

Source	Language Style / Safety Semantics	Dominant Constraint Families	Main Compiled Checks
Expert-authored	Controller-facing laboratory instructions for robot manipulation, transfer, and device use	Equipment checks, sequencing, chemical compatibility, temperature limits	Sequence enforcement, material compatibility, heat-safety checks
PubChem GHS	Short hazard / precaution statements about exposure, PPE, storage, and fire risk	Exposure, ventilation, PPE, fire safety, waste handling	Motion-limit, heat-safety, and procedure-order checks
Chemical safety reference	Curated pairwise incompatibility knowledge and reagent interaction rules	Material compatibility	Material-compatibility checks
Lab safety manual	Procedure-focused laboratory rules for transfer, cleanup, workspace, and device interaction	Sequencing, temperature, spill prevention, equipment checks, velocity limits	Sequence enforcement, heat-safety, pour-speed, and motion-limit checks

Table 8: **Source families in LabGuard-Bench.** Each family contributes rules that map to one or more implemented monitor types.

Method	Type Acc↑	Subst F1↑	Tag F1↑	Pred Soft↑	Pred Can.EM↑	Schema Valid↑	Compile Ready↑
Regex BL	70.7	95.9†	48.1†	19.9	0.8	**100.0**	**100.0**
Heuristic Mapper	19.5	95.9†	17.3	0.5	0.3	46.3	46.3
SciBERT	64.0	82.1	55.8	12.6	5.1	88.4	88.4
Qwen-7B 0-shot	68.3	85.4	63.6	20.3	7.3	95.1	95.1
Qwen3-32B 0-shot	74.4	87.1	66.8	24.7	9.8	96.3	96.3
DeepSeek-V3.1 0-shot	76.8	88.5	68.9	27.3	11.5	97.6	97.6
Qwen2.5-72B 0-shot	77.4	89.0	69.8	29.1	12.0	98.2	98.2
Qwen3-8B + LoRA	**91.5**	97.3	**93.6**	86.8	72.9	**100.0**	**100.0**
LabGuard-Hybrid	**91.4**	97.1	**93.5**	86.6	73.0	**100.0**	**100.0**

Table 9: **Random 80/20 laboratory-language grounding results.** This split is used as an in-distribution upper bound for representative grounding baselines.

Held-out source	Method	Tag F1	Pred Soft
Expert-authored (N=50)	Regex	51.1	7.4
	LoRA	72.8	11.6
	LabGuard-Hybrid	73.7	11.3
PubChem GHS (N=62)	Regex	33.6	39.7
	LoRA	61.9	28.6
	LabGuard-Hybrid	62.2	26.7
Chem. safety ref. (N=20)	Regex	45.5	80.4
	LoRA	100.0	25.5
	LabGuard-Hybrid	100.0	27.0
Lab safety manual (N=71)	Regex	51.1	5.9
	LoRA	68.1	16.8
	LabGuard-Hybrid	70.1	16.5

Table 10: **Per-fold source-held-out results.** The averages are reported in Table 2.

I.2 Executable Equivalence Testing

Synthetic states can be used to compare predicate truth values.

```
states = generate_test_states(
    rule_context)
for state in states:
    assert (eval(gold_pred, state)
        == eval(pred_pred, state))
```

I.3 Slot-Level Recovery

Slot-level recovery can separately measure substances, numeric thresholds, operators, scope tags, and directionality. This would show which predicate fields cause grounding failures.

Task	Base SR	Full SR	ΔSR	Full IR (%)	Full VR	Notes
L1 pick	94.3±4.2	95.0±3.8	+0.7	0.12%	0.3	ceiling
L1 place	92.0±5.1	93.3±4.5	+1.3	0.15%	0.2	ceiling
L1 shake	88.3±6.7	90.0±5.9	+1.7	0.18%	0.4	modest
L1 stir	85.0±7.2	87.7±6.4	+2.7	0.22%	0.5	modest
L1 open	91.7±4.8	92.3±4.2	+0.6	0.11%	0.3	ceiling
L1 close	93.3±3.9	94.0±3.5	+0.7	0.09%	0.2	ceiling
L1 pour	50.0±15.3	56.7±11.5	+6.7	0.31%	0.5	improvement
L2 HeatLiquid	33.3±10.4	46.7±11.5	+13.4	0.29%	0.4	improvement
L2 PourLiquid	45.0±15.3	48.3±14.7	+3.3	0.28%	0.5	modest
L2 ShakeBeaker	72.3±11.2	75.0±10.5	+2.7	0.19%	0.3	modest
L2 TransportBeaker	71.7±9.8	78.3±8.2	+6.6	0.24%	0.4	improvement
L3 press	96.7±3.2	98.3±2.1	+1.6	0.07%	0.2	ceiling
L3 HeatLiquid	30.0±9.5	43.3±10.8	+13.3	0.32%	0.5	improvement
L4 CleanBeaker	8.3±6.1	15.0±9.2	+6.7	0.42%	0.6	long-horizon

Table 11: **Full per-task runtime results across L1–L4 tasks.** We report success rate (SR, mean±std over 3 seeds), ΔSR (Full LabGuard minus Base), intervention rate (IR), and *assurance-layer-logged* violations per episode (VR; see Appendix C for VR semantics) under Full LabGuard, with a qualitative note column summarizing per-task behavior.

Grounding Module	Check Acc	Task F1	Ready	Catch	Prec	Safety
No grounding	34.1	28.5	0.0	16.3	45.2	60.5
Regex	75.6	68.3	100.0	82.5	88.7	73.9
LoRA (Qwen3-8B)	80.5	76.2	100.0	87.5	91.3	78.0
LabGuard-Hybrid	81.2	77.8	100.0	88.1	91.8	78.6
Gold	100.0	100.0	100.0	100.0	100.0	95.4

Table 12: Compiled-monitor replay diagnostics. We report check-function accuracy, task-scope F1, compile readiness, unsafe-scenario catch rate, precision, and aggregate safety score for monitors compiled from each grounding output.

Category	N	No Safety	Regex	LoRA	LabGuard-Hybrid	Gold
Unsafe scenarios (TP out of N):						
Incompatible Mix	10	1	6	7	7	9
Container	10	0	3	4	4	6
Temperature	10	2	5	7	7	8
Speed/Force	10	1	5	6	6	8
Order	10	2	6	7	7	9
Workspace	10	1	4	5	5	7
Spill/PPE	10	0	5	5	5	7
Total unsafe TP	70	7	34	41	41	54
Safe-control scenarios (FP out of 30):						
Safe controls	30	1	9	7	7	4

Table 13: **Stress benchmark per-category breakdown.** Unsafe rows report true positives out of 10. The safe-control row reports false positives out of 30.

Category	Definition	Count
Incompatible Mix	Mixing substances that react dangerously, such as acid + base or oxidizer + fuel	10
Temperature	Heating or cooling beyond safe limits for the substance or container	10
Workspace	Moving objects outside designated safe zones or creating collision risks	10
Container	Using the wrong container type for a substance, such as metal for acid or glass for HF	10
Speed/Force	Moving too fast near fragile objects or applying excessive force	10
Spill/PPE	Spill-prevention and PPE preconditions, such as gloves, goggles, fume hood, or splash control	10
Order	Performing steps in an unsafe sequence, such as adding acid to water versus water to acid	10
Safe-Control	Procedurally correct operations with similar surface features to unsafe scenarios	30

Table 14: Stress benchmark category definitions and counts.