SMOOTH OPERATOR: IN-PLACE SYMBOLIC PLAN REPAIR VIA CONTINUAL OPEN-WORLD PREDICATE AND OPERATOR INVENTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Open-world embodied agents frequently fail during execution because predefined symbolic planning domains are fundamentally incomplete. Existing recovery methods address unmet preconditions or planning impasses by replanning globally from scratch, causing severe plan churn. In contrast, we tackle failures where nominal preconditions are met, yet physical execution stalls and the intended action effects are not observed. We propose SMOOTH (**S**ymbolic **M**odel **O**pen-World **O**nline Repair through **T**ested **H**ypotheses), a neuro-symbolic framework for online, in-place plan repair and continual domain induction. Upon failure, SMOOTH leverages a vision-language model to jointly induce open-vocabulary relational predicates and synthesize novel operators composed from primitive skills. Candidate abstractions are physically verified via counterfactual simulation rehearsals to avoid ungrounded proposals. Verified compensatory actions are spliced directly at the failure point, preserving the valid downstream plan suffix and minimizing plan disruption. Discovered abstractions are consolidated into persistent memory for lifelong reuse. Evaluations on OPENREPAIRBENCH and MINI-BEHAVIOR show that SMOOTH achieves higher task success and domain reuse than state-of-the-art baselines, while our ablations highlight the critical roles of in-place path preservation and physical validation.

1 INTRODUCTION

Imagine preparing breakfast: you place porridge in an unfamiliar microwave, close the door, and press start. Nothing happens, because an unmodeled timer dial must be turned first. What do you do? You learn about the dial, turn it, and resume your routine rather than reconsidering every downstream decision. We expect the same resilience from an embodied robot assistant. As robots deploy into human environments, they routinely encounter objects absent from their training demonstrations or hand-engineered domains. Crucially, such gaps rarely prevent initial planning. Plan generation is inherently optimistic: missing preconditions make actions appear valid in states where they will physically fail. The planner succeeds, and the error surfaces only mid-execution. Most open-world domain expansion methods either trigger on outright planning impasses (Lu et al., 2026) or acquire abstractions during offline training (Wong et al., 2024; Li et al., 2026). We address the practical complement: repairing an executing plan that was viable under an incomplete model but fails physically at runtime.

Such failures demand local repair rather than global replanning. Standard systems discard the active plan upon failure and re-solve from scratch (Chen et al., 2024; Byrnes et al., 2025; Zhang et al., 2024), invalidating the remaining suffix and arbitrarily reshuffling downstream actions. In collaborative human-robot teams, partners coordinate around the robot's expected trajectory; silently altering the plan disrupts handovers and erodes operational trust. In contrast, localized repair resolves the failure in place, preserving downstream commitments. Under a continual learning paradigm, these repairs must not be transient patches: acquired knowledge should persist so recurring failures are avoided.

Classical plan repair preserves exactly this stability, measured by the symmetric difference D_{sym} (Fox et al., 2006), but it searches a fixed vocabulary: when the domain cannot represent the

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

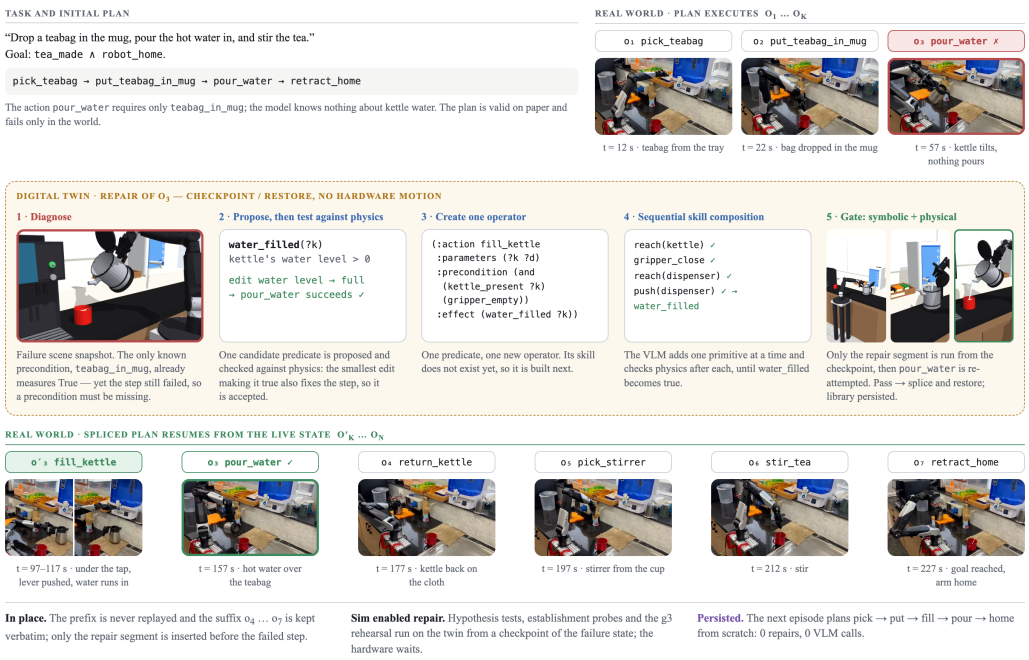


Figure 1: **A real-robot tea episode.** `pour_water` fails on an unmodeled empty kettle; the invented *kettle_filled* and composed `fill_kettle` are spliced in before it, and the next episode runs without repair.

Table 1: Qualitative comparison of our framework against representative task planning, skill discovery, and plan repair approaches. Notation: ✓ indicates full support; ✗ indicates unsupported; ○ indicates conditional, bounded, or offline support. Plan stability is measured quantitatively via D_{sym} in our experiments.

Method	Online Failure Recovery	Open-Vocabulary Predicate Disc.	Novel Operator & Skill Synth.	Memory Retention	Local In-Place Repair
DKPROMPT (Zhang et al., 2024)	✓	✗	✗	✗	✗
LASP (Chen et al., 2024)	✓	✓	✗	✗	✗
CLIMB (Byrnes et al., 2025)	✓	✓	✗	✓	✗
Ada (Wong et al., 2024)	○	✗	✓	✓	✗
Lu et al. (Lu et al., 2026)	○	✗	✓	✓	✗
PRAGMABOT (Qu et al., 2026)	✓	✗	○	✓	✗
IPyHOPPER (Zaidins et al., 2023)	✓	✗	✗	✗	✓
ReSYNC (Li et al., 2026)	○	○	✓	✓	✗
SMOOTH	✓	✓	✓	✓	✓

cause of a failure, no repair exists (Zaidins et al., 2025). Open-vocabulary methods extend the domain with foundation models, but only half of it at a time (Table 1): predicate learners keep a fixed action set and replan globally (Chen et al., 2024; Byrnes et al., 2025), while operator learners assume the predicates already exist (Wong et al., 2024; Lu et al., 2026). To our knowledge, no framework induces both predicates and executable operators online from execution failures while preserving the executing plan.

To address these limitations, we propose an autonomous framework for in-place symbolic plan repair via continual open-vocabulary predicate and operator induction. Our contributions are:

- **Online In-Place Plan Repair.** We eliminate global replanning churn by splicing compensatory actions directly at the point of failure and verifying causal consistency across the remaining suffix, preserving collaborative commitments.

- **Failure-Driven Predicate Discovery.** When an action fails despite satisfied preconditions, a vision-language model diagnoses the missing condition to induce an open-vocabulary predicate without relying on a predefined lexicon.
- **Operator Synthesis via Skill Chaining.** To achieve the newly induced predicate, the framework generates a PDDL operator schema and realizes its continuous execution by chaining existing primitive skills, avoiding sample-inefficient RL.
- **Persistent Abstraction Memory.** Discovered predicates, grounding classifiers, and chained operators are consolidated into long-term memory for zero-shot reuse across future tasks.

2 Related Work

Plan repair and domain model repair. Execution monitoring decides whether a failure matters for the remaining plan (Fikes et al., 1972; Wilkins, 1985; Fritz & McIlraith, 2007). Plan repair then modifies the existing plan rather than replanning, finding solutions faster and with fewer revisions (Fox et al., 2006; van der Krogt & de Weerdt, 2005; Gerevini & Serina, 2010; Saetti & Scala, 2022), including in hierarchical task networks (Goldman et al., 2020; Zaidins et al., 2023; Höller et al., 2020). Domain model repair instead edits preconditions and effects to fit observed plans, offline or online (Lin et al., 2023; Gragera et al., 2023; Bercher et al., 2025; Lamanna et al., 2021). All of these assume a fixed vocabulary of predicates and operators; we share their goal of preserving the plan but also extend the vocabulary.

Learning predicates and operators. Predicate invention learns new state abstractions offline from demonstrations (Silver et al., 2023; Li et al., 2025; Athalye et al., 2026; Liang et al., 2025) or interactively from failures and language feedback (Han et al., 2024; Chen et al., 2024; Byrnes et al., 2025), but keeps a fixed action set. Operator synthesis fills missing actions via reinforcement learning after an impasse (Sarathy et al., 2021; Goel et al., 2022; Lorang et al., 2024), LLM-proposed schemas (Wong et al., 2024; Lu et al., 2026; Li et al., 2024), or recovery policies (Vats et al., 2024), but assumes the predicates exist. ReSYNC (Li et al., 2026) and PDDLLM (Huang et al., 2026) induce both, but offline, from a failure curriculum or demonstrations, and deploy a frozen library; we induce both online from a single execution failure.

Vision-language models for failure recovery. VLMs detect and explain robot failures (Liu et al., 2023; Duan et al., 2025; Guo et al., 2024), including by checking PDDL preconditions and effects against images (Zhang et al., 2024; Oloo et al., 2026). Recovery then draws on a fixed skill set (Sundara Raman et al., 2024; Joublin et al., 2024), a predefined ontology of strategies (Cornelio & Diab, 2024), or sub-goals of the original plan (Kalithasan et al., 2024), and lessons are retained through reflection or retrieval (Shinn et al., 2023; Qu et al., 2026). We likewise use a VLM to diagnose failures, but use the diagnosis to revise the domain model itself, stored as reusable symbols rather than instance-specific experiences.

3 Problem Formulation

Setting. The robot acts in a checkpointable simulator with state $x \in \mathcal{X}$. A fixed function ψ extracts a *scene* $s = \psi(x)$ over named *dimensions*: typed objects $\mathcal{O}$ (poses, extents), articulated controls $\mathcal{J}$, and latent readings $\mathcal{C}$ no control exposes, plus what the gripper holds. It can also *edit* any dimension and let its dynamics settle — exposing what can change, never which change fixes a failure. Skills $\kappa \in \mathcal{K}$ (engineered named skills $\mathcal{K}_{\mathrm{N}}$ or composed on demand from primitives Π, §4.3) return state and a success flag $(x', b) = \mathrm{Exec}(\kappa, x)$ with $b \in \{0, 1\}$.

Model. SMOOTH learns $\mathcal{M} = \langle \mathcal{P}, \mathcal{A} \rangle$. Each $p \in \mathcal{P}$ carries an executable classifier φ_p over the scene, so symbolic state is *measured*, never believed: $\mathrm{Abs}_{\mathcal{P}}(s) = \{\, p(\bar{o}) : \varphi_p(s, \bar{o}) = 1 \,\}$. Each $a = \langle \mathrm{pre}_a, \mathrm{add}_a, \mathrm{del}_a, \kappa_a \rangle \in \mathcal{A}$ is a STRIPS action over $\mathcal{P}$ naming its implementing skill $\kappa_a \in \mathcal{K}$; types restrict argument grounding. The seed $\mathcal{M}_0 = \langle \mathcal{P}_0, \mathcal{A}_0 \rangle$ plans but is incomplete, as deployed models are.

Task and failure. A task specifies an instruction ℓ, initial state $x_0 \in \mathcal{X}$, goal G, and a deployment context D describing the world (never a failure mode or fix), attempted over repeated episodes. A

planner returns $\rho = \langle a_1, \ldots, a_n \rangle$ from $\mathrm{Abs}_{\mathcal{P}}(\psi(x_0))$ to G, executed skill by skill. Step a_k *fails* at state x_f ($s_f = \psi(x_f)$) when its skill fails ($b = 0$) though judged applicable. Evaluating its preconditions on s_f diagnoses the failure as: MISSING (all hold, yet it failed) or UNESTABLISHED (one is false and no operator establishes it).

Repair problem. Find $\mathcal{M} \to \mathcal{M}'$ and a repair segment ρ^r such that (R1) ρ^r then a re-attempt of a_k succeeds; (R2) $\rho_{<k} \oplus \rho^r \oplus \rho_{\geq k}$ — the repair before the *untouched* original suffix — stays valid under $\mathcal{M}'$ and reaches G; and (R3) later episodes plan and execute with $\mathcal{M}'$ alone, without failure. Convergence: episodes until zero repairs and zero VLM calls.

4 METHOD

4.1 OVERVIEW

In each episode, the robot measures its initial symbolic state, plans with its current model $\mathcal{M}$, and executes ρ step by step. When a step a_k fails physically, SMOOTH diagnoses the failure by measuring its known preconditions on the failure scene s_f (full loop and pseudocode in Appendix A).

For a MISSING precondition, SMOOTH executes a three-stage repair:

1. *Predicate invention* (§4.2): the VLM $\mathcal{V}$ proposes rival executable predicates φ_h, which are physically tested via minimal directed scene interventions.
2. *Operator synthesis* (§4.3): an operator establishing the winning predicate is found by reusing a library operator, binding a named skill, or composing a sequence of primitives, verified via physical probes.
3. *Local gating and splicing* (§4.4): the repair segment ρ^r is spliced before a_k. A symbolic check ensures the untouched plan suffix still achieves G, and a checkpointed physical rehearsal verifies ρ^r and a_k. On success, the repair is committed, the episode resumes, and $\mathcal{M}'$ is persisted; otherwise, changes are rolled back.

While MISSING preconditions require this complete pipeline, an UNESTABLISHED failure directly reuses its later stages: it already names the target predicate, bypassing predicate invention to trigger operator synthesis directly. We therefore focus the subsequent exposition on the complete MISSING repair loop.

4.2 PREDICATE INVENTION BY PHYSICAL HYPOTHESIS TESTING

When a step fails despite satisfied known preconditions (a MISSING diagnosis, e.g., `pour_water` failing because the kettle is unmodeled as empty), SMOOTH induces the missing predicate via physical hypothesis testing.

Proposing and screening rival hypotheses. Given the failure scene $s_f = \psi(x_f)$, image, and task context, the VLM proposes 2–4 rival predicates $h = \langle \bar{\tau}_h, \bar{o}_h, \varphi_h \rangle$ as executable scene classifiers $\varphi_h(s, \bar{o})$, while unattached library predicates join the pool for reuse. Programs raising runtime errors on s_f are rejected, and multi-dimensional conjunctions are split into atomic predicates.

Physical sufficiency via directed interventions. A valid candidate must evaluate false at failure ($\varphi_h(s_f, \bar{o}) = 0$). Rather than sweeping arbitrary scenes, SMOOTH targets only the dimensions $\mathrm{dims}(\varphi_h)$ read by the classifier (joints, latents, or object removals). Applying the minimal satisfying edit set $E_h^\star \subseteq \mathcal{I}_h$ to a checkpoint of x_f, we re-attempt the failed skill κ_{a_k}. A hypothesis is *physically sufficient* iff the step now succeeds:

$$\begin{aligned} \textsc{Sufficient}(h) \iff \varphi_h(s_f, \bar{o}) = 0 \ \wedge \ \exists E_h^\star \subseteq \mathcal{I}_h : \\ \varphi_h(\psi(x_{E_h^\star}), \bar{o}) = 1 \ \wedge \ \mathrm{Exec}(\kappa_{a_k}, x_{E_h^\star}) = (\cdot, 1). \end{aligned} \tag{1}$$

Disambiguation and attachment. When multiple rivals are sufficient, SMOOTH breaks ties via causal entailment and physical achievability: (i) h asymmetrically entails h' if $E_h^\star$ also satisfies $\varphi_{h'}$, pruning extraneous proxy edits (e.g., moving a tray to move a cup); and (ii) for independent candidates, SMOOTH tests whether an available library skill $\mathcal{K}_{\mathrm{N}}$ can actually bring about the state,

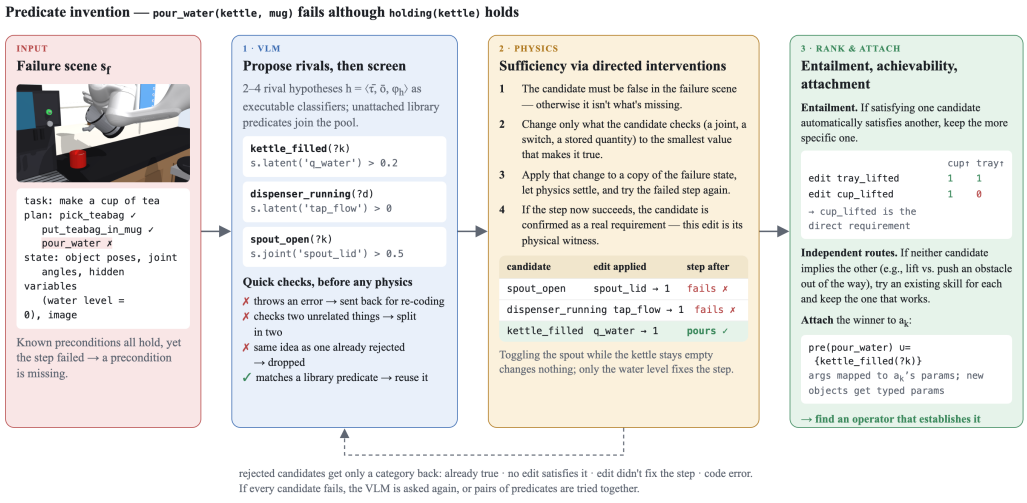


Figure 2: **Predicate invention.** The VLM proposes candidate predicates that are physically verified via AST-directed scene edits and ranked by skill achievability to isolate the true missing constraint.

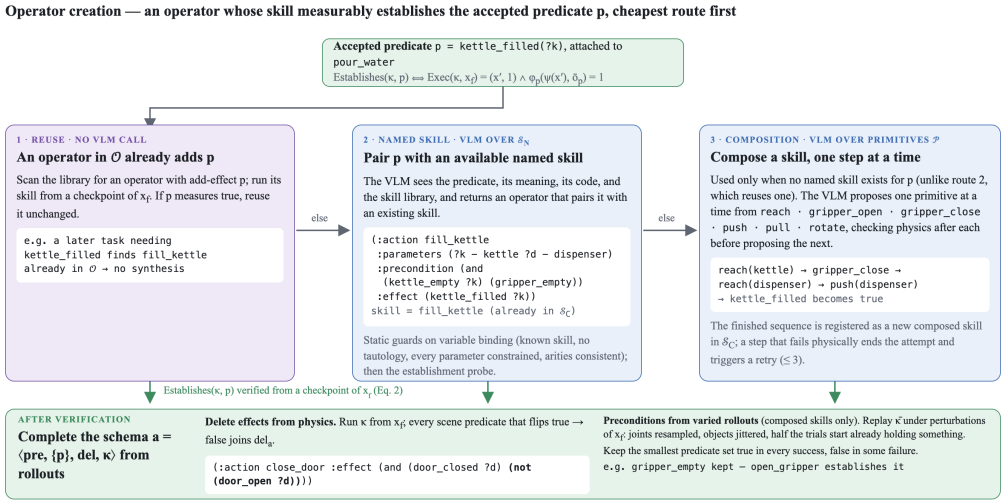


Figure 3: **Operator creation.** For an induced predicate p, SMOOTH evaluates existing operators, queries a named skill, or composes primitives via step-by-step physical rollouts, fitting delete effects and preconditions from simulation.

selecting the actionable hypothesis. The winning predicate is attached to the operator ($\text{pre}_{a_k} \leftarrow \text{pre}_{a_k} \cup \{p(\bar{v})\}$), widening its signature if new objects are referenced.

4.3 Operator Creation and Skill Composition

An induced precondition $p(\bar{o}_p)$ requires an operator a whose controller κ verifiably establishes it in physics from the failure state:

$$\textsc{Establishes}(\kappa, p) \iff \text{Exec}(\kappa, x_f) = (x', 1) \ \wedge \ \varphi_p(\psi(x'), \bar{o}_p) = 1. \tag{2}$$

Hierarchical skill discovery. Candidate skills are searched in increasing order of complexity, each checked with Eq. 2. *Reuse* tries existing operators in $\mathcal{A}$ that establish p, or chains of up to 3. Failing that, the VLM maps p to a *named skill* $\kappa \in \mathcal{K}_{\text{N}}$ (e.g., *kettle_filled* to `fill_kettle`). Otherwise

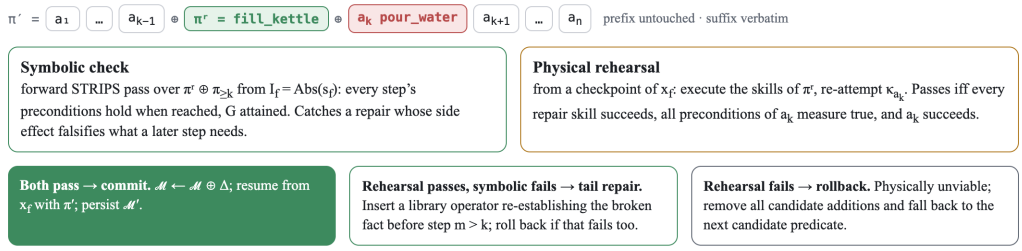


Figure 4: **Local gate and splicing.** The verified repair segment ρ^r is spliced before the failed step a_k, preserving the remaining plan suffix verbatim without global replanning.

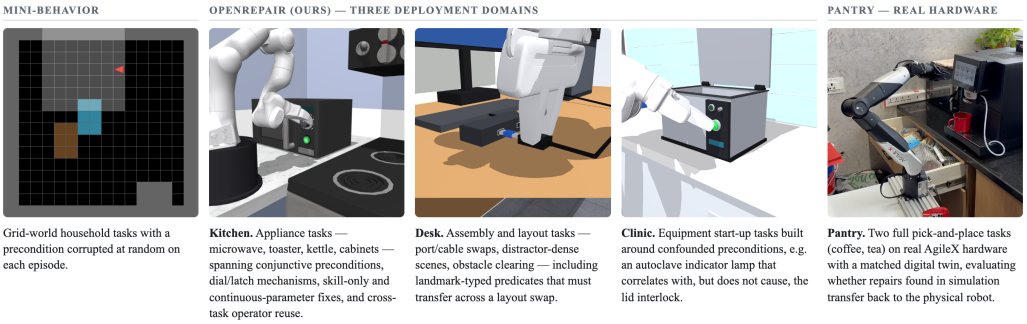


Figure 5: **Evaluation suites.** MINI-BEHAVIOR grid-world household tasks, the three OPENREPAIRBENCH domains (kitchen, study desk, clinic), and real-robot pantry tasks repaired on a digital twin.

it *composes primitives* $\bar{\kappa} = \langle \pi_1, \ldots, \pi_m \rangle \in \Pi$, which SMOOTH rolls out step by step against physics, feeding intermediate results back to repair broken sequences until p holds.

Grounding schema effects and preconditions. To complete $a = \langle \mathrm{pre}_a, \{p\}, \mathrm{del}_a, \kappa \rangle$, running κ from x_f reveals side effects: every predicate that flips from true to false joins del_a (e.g., closing a door deletes *door_open*). Composed skills have no predefined preconditions, so SMOOTH perturbs x_f and keeps the minimal set of predicates that separates successful executions from failures (e.g., *gripper_empty*).

4.4 LOCAL ACCEPTANCE, SPLICING, AND PERSISTENCE

Rather than replanning globally, SMOOTH splices the minimal repair segment ρ^r directly before the failed step, preserving the original suffix: $\rho' = \rho_{<k} \oplus \rho^r \oplus \rho_{\geq k}$. Two checks run before execution. A **symbolic suffix walk**, a forward STRIPS pass over $\rho^r \oplus \rho_{\geq k}$, ensures that side effects neither clobber downstream preconditions nor invalidate G. A **physical delta rehearsal** from a checkpoint of x_f confirms that ρ^r executes and lets the failed step a_k succeed. If both pass, SMOOTH **commits**: ρ' resumes in place and the new predicates and operators are consolidated into $\mathcal{M}$ for reuse in later tasks. If rehearsal passes but a downstream precondition at step $m > k$ is clobbered, a **tail repair** inserts a known operator restoring it just before step m. Otherwise, the repair is **rolled back** and the next hypothesis is tried.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate SMOOTH on two benchmarks (Figure 5): our OPENREPAIRBENCH, comprising 77 long-horizon manipulation tasks across Kitchen, Study Desk, and Clinic domains powered by a

Table 2: **Predicate induction.** Top: OPENREPAIRBENCH, 50 tasks (68 episodes) by failure group. Bottom: MINI-BEHAVIOR, 16 corrupt-precondition cases (48 episodes), all with directly visible conditions. Calls are average VLM queries per episode.

	SMOOTH				LASP (Chen et al., 2024)				CLIMB (Byrnes et al., 2025)			
Failure Group	**Succ.**	**Reuse**	$D_{\text{sym}}\downarrow$	**Calls**$\downarrow$	**Succ.**	**Reuse**	$D_{\text{sym}}\downarrow$	**Calls**$\downarrow$	**Succ.**	**Reuse**	$D_{\text{sym}}\downarrow$	**Calls**$\downarrow$
Group 1 (Direct)	**100.0%**	**100.0%**	**1.52**	3.7	33.3%	0.0%*	1.62	14.2	36.0%†	40.0%	1.60	**3.2**
Group 2 (Masked)	**100.0%**	**72.0%**	1.33	4.5	66.7%	0.0%*	1.24	13.3	25.0%†	52.0%	**1.00**	**3.4**
Group 3 (Multi-Gate)	**100.0%**	**75.0%**	**2.77**	5.8	0.0%	0.0%*	2.90	31.4	0.0%†	0.0%	—	**3.8**
Total / Average	**100.0%**	**85.8%**	1.85	4.5	37.8%	0.0%*	1.91	19.1	22.6%†	30.9%	**1.39**	**3.4**

MINI-BEHAVIOR	Succ.	Reuse	$D_{\text{sym}}\downarrow$	Calls$\downarrow$
SMOOTH	**70.0%**	**30.0%**	**1.0**	**2.0**
LASP (Chen et al., 2024)	2.0%	n/a*	8.0	9.9
CLIMB (Byrnes et al., 2025)	16.0%	6.0%	4.6	12.4

*LASP maintains no lifelong memory across tasks; learned predicates are discarded after each episode, yielding 0.0% cross-task reuse.
†CLIMB coverage on benchmark subset.

cuRobo-backed digital twin, and MINI-BEHAVIOR for multi-room relational tasks. Across 3-turn lifelong deployments ($T_{\max} = 3$), we benchmark against re-implementations of LASP (Chen et al., 2024) and CLIMB (Byrnes et al., 2025) for predicate induction, Lu et al. (2026) for operator synthesis, and a global replanning baseline for compound failure modes, using Qwen3-VL and GPT-4-class VLM backbones stated per experiment. Tasks are categorized into three failure tiers: **Group 1** (directly observable single gates, e.g., joint limits or gripper states), **Group 2** (opaque single conditions, including out-of-span latent scalars, negative polarities, and external relational dependencies), and **Group 3** (jointly masked multi-precondition bottlenecks such as coupled appliance latches and timers). Performance is assessed via task **Success Rate** (%), plan churn via symmetric difference **Plan Preservation** ($D_{\text{sym}} = |\pi \setminus \pi'| + |\pi' \setminus \pi|$), and **Knowledge Reuse** (%), defined as zero-shot execution from persistent memory without triggering VLM re-repair.

5.2 PREDICATE INDUCTION

Results. On OPENREPAIRBENCH (Table 2), SMOOTH solves every task, reuses its learned predicates on 85.8% of later turns, and needs 4.2× fewer VLM queries than LASP (4.5 vs. 19.1). The baselines hold up only on directly observable faults. On Group 2, CLIMB drops to 25.0% by misattributing failures to the robot's posture, and LASP to 66.7%; on Group 3, where several gates are masked at once, neither solves a task, and LASP spends 31.4 calls per episode replanning. On MINI-BEHAVIOR, SMOOTH achieves 70.0% success with 2.0 VLM calls per episode, avoiding the costly open-loop replanning of LASP (2.0%, 9.9 calls) and CLIMB (16.0%, 12.4 calls). Its longer tasks also expose the plan-preservation gap that our primary benchmark's short horizons compress: local splicing preserves the valid suffix ($D_{\text{sym}} = 1.0$), whereas replanning from scratch diverges ($D_{\text{sym}} \geq 4.6$).

5.3 AUTONOMOUS OPERATOR SYNTHESIS

Here the vocabulary is complete, but an operator needed to establish one of its conditions is withheld from $\mathcal{O}_0$. We compare against Lu et al. (2026), which queries the LLM only when its planner finds no plan, proposes the missing operator with a policy realizing it, and keeps both for later tasks.

Results. SMOOTH completes every failure group (Table 3), while Lu et al. (2026) reaches 68.3%, dropping on Group 2 (58.3%) and Group 3 (64.3%). Both methods keep what they learn, so reuse tracks first-turn success. The baseline commits to an operator's symbolic specification before touching the scene, and 10 of its 18 first-turn failures propose effects no skill sequence can realize; SMOOTH derives the operator from a physically rehearsed witness. Both insert the operator into an otherwise unchanged plan, so churn is low for both ($D_{\text{sym}} = 1.17$ vs. 1.12). We do not evaluate operator synthesis or compound repair on MINI-BEHAVIOR: withholding any of its operators leaves no valid initial plan, so the failure is a planning impasse before execution rather than the execution failure SMOOTH targets.

Table 3: **Operator synthesis** on OPENREPAIRBENCH. SMOOTH is evaluated on 15 tasks (31 episodes) and Lu et al. on 52 tasks (104 episodes). Both methods use gpt-4o. An episode is one turn; calls are average VLM queries per episode.

	SMOOTH				Lu et al. (2026)			
Failure Group	**Succ.**	**Reuse**	$D_{\text{sym}}\downarrow$	**Calls**$\downarrow$	**Succ.**	**Reuse**	$D_{\text{sym}}\downarrow$	**Calls**$\downarrow$
Group 1 (Direct)	**100.0%**	**90.0%**	**1.00**	**1.2**	75.0%	73.1%	1.05	2.3
Group 2 (Masked)	**100.0%**	**100.0%**	**1.00**	**1.5**	58.3%	50.0%	**1.00**	3.8
Group 3 (Multi-Gate)	**100.0%**	**100.0%**	3.00	**5.0**	64.3%	57.1%	**1.33**	6.4
Total / Average	**100.0%**	**92.3%**	1.17	**1.5**	68.3%	63.5%	**1.12**	3.8

Table 4: **Compound failure recovery** on OPENREPAIRBENCH. SMOOTH is evaluated on 8 anchor scenarios $\times$ 3 seeds (24 episodes; best configuration over gpt-4.1, gpt-4o and gpt-4o-mini) and global replanning on 26 tasks (Qwen3-VL-8B-Thinking). Success counts an episode only if every turn reaches G; calls are average VLM queries per turn.

	SMOOTH			Global Replanning		
Failure Group	**Succ.**	**Reuse**	**Calls**$\downarrow$	**Succ.**	**Reuse**	**Calls**$\downarrow$
Group 1 (Direct)	**100.0%**	**80.0%**	**1.0**	75.0%	62.5%	1.1
Group 2 (Masked)	**88.9%**	**88.9%**	2.2	80.0%	80.0%	**0.9**
Group 3 (Multi-Gate)	**100.0%**	**100.0%**	1.2	80.0%	80.0%	**0.8**
Total / Average	**95.8%**	**85.2%**	1.4	76.9%	69.2%	**1.0**

5.4 COMPOUND FAILURE RECOVERY

Here both the predicate and the operator that establishes it are withheld. For SMOOTH we report the best configuration from the ablation study (Section 5.5). The baseline, global replanning, has the VLM regenerate the full plan from the failure feedback and stores successful plans in a retrieval memory that later turns of the same task replay without a query.

Results. SMOOTH solves 95.8% of compound episodes against 76.9% for global replanning (Table 4). Replanning uses fewer queries per turn (1.0 vs. 1.4), but its memory stores plans, not domain knowledge, so it helps only when a solved task recurs. SMOOTH writes the new predicate and operator into the domain: 85.2% of later turns need no query (vs. 69.2%), and the cross-task *transfer* scenario reuses the stored operator on a new task with a single query.

5.5 ABLATION STUDY

We ablate the plan update, physical rehearsal, and the VLM backbone on 8 OPENREPAIRBENCH anchor scenarios spanning all three failure groups, with 3 seeds each, drawn from ten sweeps (1,366 episodes) over three GPT backbones. For each scenario we report the best configuration of SMOOTH and, to guard against selection bias, the unselected pooled numbers (protocol in Appendix F).

In-place repair vs. global replanning. A Replan arm receives exactly the knowledge SMOOTH acquired: the learned library is cut back to its end-of-turn state, and on an execution failure pyperplan replans from the live state to G with no VLM call. Sharing one model, the arms succeed equally by construction (95.8% each; Table 5) but differ in plan stability and in robustness to imperfect learned models. On *obstacle* (Group 2), the repair widened `place_widget_on_pad` with a new `?pad` argument; SMOOTH keeps the suffix verbatim ($D_{\text{sym}} = 1$), whereas replanning re-grounds the downstream step ($D_{\text{sym}} = 3$). Over all 1,327 comparable episodes, paired turn-1 outcomes favour SMOOTH 7 to 0: each time the learned model was subtly wrong and the planner trusted it, while SMOOTH executed the segment it had physically rehearsed (Appendix C.4). Pooled D_{sym}

Table 5: **In-place repair vs. global replanning.** Both arms use the predicates and operators SMOOTH learned, reconstructed per turn; Replan calls pyperplan from the live failure state to G instead of splicing. D_{sym} on turn-1 successes. Top: best configuration per scenario (3 seeds). Bottom: all 1,327 comparable episodes.

	Success		$D_{\text{sym}}\downarrow$	
Failure Group	**SMOOTH**	**Replan**	**SMOOTH**	**Replan**
Group 1 (Direct)	100.0%	100.0%	1.00	1.00
Group 2 (Masked)	88.9%	88.9%	**1.00**	1.50
Group 3 (Multi-Gate)	100.0%	100.0%	2.00	2.00
Total / Average	95.8%	95.8%	**1.13**	1.30
Pooled success	54.5%	54.4%	—	—
Pooled turn-1 goal	**55.8%**	55.2%	—	—

Table 6: **Effect of physical rehearsal.** Hypothesis testing, establishment probes, and delta rehearsal removed (symbolic suffix check kept); 26 OPENREPAIRBENCH tasks, turn 1, Qwen3-VL-8B. ‡Composed operator, distractor, numeric threshold, abstention tie, unreliable skill, two repairs in one turn.

Failure type	**SMOOTH**	**No physics**
Conjunctive	**4/4**	0/4
Confounded	**2/2**	0/2
Out-of-span latent	**5/5**	3/5
Other physics-relevant‡	**6/6**	2/6
Control	**9/9**	2/9
Total	**100.0%**	26.9%
Accepted but invalid	**0**	19
VLM calls / run ↓	**3.92**	4.19

is slightly higher for SMOOTH (1.46 vs. 1.32), an artifact of repair segments that instantiate every ground instance of an invented operator.

Physical rehearsal. Table 6 disables every physical check while keeping the symbolic suffix walk. Without physics, the VLM's hypotheses can no longer be tested, and the gate accepts a repair in all 26 runs. However, 19 of these repairs are invalid: the re-attempted step still fails, and turn-1 success collapses from 100% to 26.9%. In 25 of 26 runs, the ablated system admits at least one repair that rehearsal would have refused. The degradation is sharpest on conjunctive and confounded failures ($0/6$), where the VLM's most plausible explanation is incomplete or a proxy. Even control tasks degrade ($2/9$), because an unverified operator is committed whenever its symbolic effects look right. Removing physics does not save queries either (4.19 vs. 3.92 calls), since rejected proposals are replaced by equally unverified ones.

VLM backbone. SMOOTH degrades gracefully with a weaker VLM: under the best configuration, gpt-4.1, gpt-4o and gpt-4o-mini succeed on 87.5%, 83.3% and 70.8% of episodes, and physical testing rejects most of the weakest model's incorrect predicates before they reach the domain (Appendix C.5).

6 CONCLUSION

We presented SMOOTH, which repairs an executing plan in place when a step fails despite its modeled preconditions holding. A vision-language model proposes the missing predicate and the operator that establishes it, physical hypothesis testing on a digital twin accepts only proposals whose effects are observed, and the verified segment is spliced before the failed step while the rest of the plan is kept. The learned predicates and operators persist in the domain model, so later tasks plan with them directly. Across predicate induction, operator synthesis, and compound failures on OPENREPAIRBENCH, SMOOTH solves more tasks and reuses more of what it learns than the baselines, and removing physical validation drops turn-1 success from 100% to 26.9%. SMOOTH relies on a checkpointable twin, and its downstream check is symbolic; validating learned models on the real robot is the main direction for future work (Appendix G).

AI USE STATEMENT

We have not used generative AI tools to develop hypotheses, theoretical models or conceptual frameworks, or to design the methodology and experiments; generating synthetic data, mathematical claims and proofs, translation, dataset cleaning and qualitative analysis are not applicable to this work. Additionally, we used generative AI tools to identify and summarize related literature, to draft sections of the paper and polish its writing (rephrasing, shortening text to fit the page limit, and formatting tables), to create and edit figures, and to format references. We have reviewed all

AI-assisted work: the authors checked and edited every AI-drafted passage, checked every reported number against our experiment logs, and checked every AI-suggested reference against its published version. The vision-language models that SMOOTH and the baselines call at run time are components of the evaluated methods, not tools used to write the paper; Section 5 and Appendix C.5 state which model each experiment uses. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Ashay Athalye, Nishanth Kumar, Tom Silver, Yichao Liang, Jiuguang Wang, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. From pixels to predicates: Learning symbolic world models via pretrained VLMs. *IEEE Robotics and Automation Letters*, 11(4):4002–4009, 2026.

Pascal Bercher, Sarath Sreedharan, and Mauro Vallati. A survey on model repair in AI planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 10371–10380, 2025.

Walker Byrnes, Miroslav Bogdanovic, Avi Balakirsky, Stephen Balakirsky, and Animesh Garg. CLIMB: Language-guided continual learning for task planning with iterative model building. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

Guanqi Chen, Lei Yang, Ruixing Jia, Zhe Hu, Yizhou Chen, Wei Zhang, Wenping Wang, and Jia Pan. Language-augmented symbolic planner for open-world task planning. In *Robotics: Science and Systems (RSS)*, 2024.

Cristina Cornelio and Mohammed Diab. RECOVER: A neuro-symbolic framework for failure detection and recovery. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12435–12442, 2024. doi: 10.1109/IROS58592.2024.10801853.

Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. AHA: A vision-language-model for detecting and reasoning over failures in robotic manipulation. In *International Conference on Learning Representations (ICLR)*, 2025.

Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3:251–288, 1972.

Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 212–221. AAAI Press, 2006.

Christian Fritz and Sheila A. McIlraith. Monitoring plan optimality during execution. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2007.

Alfonso Gerevini and Ivan Serina. Efficient plan adaptation through replanning windows and heuristic goals. *Fundamenta Informaticae*, 102(3–4):287–323, 2010.

Shivam Goel, Yash Shukla, Vasanth Sarathy, Matthias Scheutz, and Jivko Sinapov. RAPid-Learn: A framework for learning to recover for handling novelties in open-world environments. In *IEEE International Conference on Development and Learning (ICDL)*, pp. 15–22, 2022.

Robert P. Goldman, Ugur Kuter, and Richard G. Freedman. Stable plan repair for state-space HTN planning. In *Working Notes of the ICAPS Workshop on Hierarchical Planning (HPlan)*, 2020.

Alba Gragera, Raquel Fuentetaja, Ángel García-Olaya, and Fernando Fernández. A planning approach to repair domains with incomplete action effects. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 33, pp. 153–161, 2023.

Yanjiang Guo, Yen-Jen Wang, Lihan Zha, and Jianyu Chen. DoReMi: Grounding language model by detecting and recovering from plan-execution misalignment. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12124–12131, 2024.

Muzhi Han, Yifeng Zhu, Song-Chun Zhu, Ying Nian Wu, and Yuke Zhu. InterPreT: Interactive predicate learning from language feedback for generalizable task planning. In *Robotics: Science and Systems (RSS)*, 2024.

Daniel Höller, Pascal Bercher, Gregor Behnke, and Susanne Biundo. HTN plan repair via model transformation. In *KI 2020: Advances in Artificial Intelligence*, volume 12325 of *Lecture Notes in Computer Science*, pp. 88–101. Springer, 2020.

Jinbang Huang, Yixin Xiao, Zhanguang Zhang, Mark Coates, Jianye Hao, and Yingxue Zhang. One demo is all it takes: Planning domain derivation with LLMs from a single demonstration. In *International Conference on Learning Representations (ICLR)*, 2026.

Frank Joublin, Antonello Ceravola, Pavel Smirnov, Felix Ocker, Joerg Deigmoeller, Anna Belardinelli, Chao Wang, Stephan Hasler, Daniel Tanneberg, and Michael Gienger. CoPAL: Corrective planning of robot actions with large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8664–8670, 2024.

Namasivayam Kalithasan, Arnav Tuli, Vishal Bindal, Himanshu Gaurav Singh, Parag Singla, and Rohan Paul. Learning to recover from plan execution errors during robot manipulation: A neuro-symbolic approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12632–12639, 2024.

Danial Kamali and Parisa Kordjamshidi. NePTune: A neuro-pythonic framework for tunable compositional reasoning on vision-language. In *International Conference on Learning Representations (ICLR)*, 2026.

Leonardo Lamanna, Alessandro Saetti, Luciano Serafini, Alfonso Gerevini, and Paolo Traverso. Online learning of action models for PDDL planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4112–4118, 2021. doi: 10.24963/ijcai.2021/566.

Bowen Li, Tom Silver, Sebastian Scherer, and Alexander Gray. Bilevel learning for bilevel planning. In *Robotics: Science and Systems (RSS)*, 2025.

Bowen Li, Mayank Mishra, Y. Isabel Liu, Stone Tao, Nishanth Kumar, Alexander Gray, Ruwan Wickramarachchi, Jonathan Francis, Sebastian Scherer, and Tom Silver. Recover, discover, plan: Learning skills and concepts from robot failures. *arXiv preprint arXiv:2606.18328*, 2026.

Zhaoyi Li, Kelin Yu, Shuo Cheng, and Danfei Xu. LEAGUE++: Empowering continual robot learning through guided skill acquisition with large language models. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024.

Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B. Tenenbaum, Tom Silver, João F. Henriques, and Kevin Ellis. VisualPredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. In *International Conference on Learning Representations (ICLR)*, 2025.

Songtuan Lin, Alban Grastien, and Pascal Bercher. Towards automated modeling assistance: An efficient approach for repairing flawed planning domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing robot experiences for failure explanation and correction. In *Conference on Robot Learning (CoRL)*, pp. 3468–3484, 2023.

Pierrick Lorang, H. Horvath, T. Kietreiber, P. Zips, C. Heitzinger, and Matthias Scheutz. Adapting to the "open world": The utility of hybrid hierarchical reinforcement learning and symbolic planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 508–514, 2024.

Hong Lu, Pierrick Lorang, Timothy R. Duggan, Jivko Sinapov, and Matthias Scheutz. Novelty adaptation through hybrid large language model (LLM)-symbolic planning and LLM-guided reinforcement learning. *arXiv preprint arXiv:2603.11351*, 2026.

Austine Oloo, Zainab Altaweel, Yohei Hayamizu, Peiqi Liu, Yan Ding, Saeid Amiri, Hao Yang, Andy Kaminski, Chad Esselink, Chris Paxton, Xiaohan Zhang, and Shiqi Zhang. Robot planning and situation handling with active perception. *arXiv preprint arXiv:2604.26988*, 2026.

Kaixian Qu, Guowei Lan, René Zurbrügg, Changan Chen, Christopher E. Mower, Haitham Bou-Ammar, and Marco Hutter. A pragmatist robot: Learning to plan tasks by experiencing the real world. *IEEE Robotics and Automation Letters*, 11(5):5606–5613, 2026. doi: 10.1109/LRA.2026.3669804.

Alessandro Saetti and Enrico Scala. Optimising the stability in plan repair via compilation. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, volume 32, pp. 316–320, 2022. doi: 10.1609/icaps.v32i1.19815.

Vasanth Sarathy, Daniel Kasenberg, Shivam Goel, Jivko Sinapov, and Matthias Scheutz. SPOTTER: Extending symbolic planning operators through targeted reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1118–1126, 2021.

Sarvam AI. Sarvam-105b. `https://www.sarvam.ai`, 2026. Accessed through the OpenAI-compatible API, September 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 8634–8652, 2023.

Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Joshua B. Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 12120–12129, 2023.

Shreyas Sundara Raman, Vanya Cohen, Ifrah Idrees, Eric Rosen, Raymond Mooney, Stefanie Tellex, and David Paulius. CAPE: Corrective actions from precondition errors using large language models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 14070–14077, 2024.

TypeSafe AI. Jev: A system one model for typed, calibrated judgments. `https://docs.typesafe.ai`, 2026. Model `jev-1.13.0`, accessed September 2026.

Roman van der Krogt and Mathijs de Weerdt. Plan repair as an extension of planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 161–170, 2005.

Shivam Vats, Devesh K. Jha, Maxim Likhachev, Oliver Kroemer, and Diego Romeres. RecoveryChaining: Learning local recovery policies for robust manipulation. *arXiv preprint arXiv:2410.13979*, 2024.

David E. Wilkins. Recovering from execution errors in SIPE. *Computational Intelligence*, 1(1): 33–45, 1985.

Lionel Wong, Jiayuan Mao, Pratyusha Sharma, Zachary S. Siegel, Jiahai Feng, Noa Korneev, Joshua B. Tenenbaum, and Jacob Andreas. Learning adaptive planning representations with natural language guidance. In *International Conference on Learning Representations (ICLR)*, 2024.

Paul Zaidins, Mark Roberts, and Dana Nau. Implicit dependency detection for HTN plan repair. In *Proceedings of the ICAPS Workshop on Hierarchical Planning (HPlan)*, 2023.

Paul Zaidins, Robert P. Goldman, Ugur Kuter, Dana Nau, and Mark Roberts. HTN plan repair algorithms compared: Strengths and weaknesses of different methods. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2025.

Xiaohan Zhang, Zainab Altaweel, Yohei Hayamizu, Yan Ding, Saeid Amiri, Hao Yang, Andy Kaminski, Chad Esselink, and Shiqi Zhang. DKPROMPT: Domain knowledge prompting vision-language models for open-world planning. *arXiv preprint arXiv:2406.17659*, 2024.

APPENDIX CONTENTS

A Method details: overview, pseudocode, scene interface, diagnosis, predicate invention, operator synthesis, local gate, persistence, ablation flags
B Prompts
C Additional ablations: prompt engineering, why hypotheses are rejected, flag configurations, repair vs. replan (extended), VLM backbone (extended)
D Evaluating learned predicates on an imperfect twin or a real robot: Jev and NePTune-style soft programs
E Learned abstractions
F Benchmark and evaluation protocol
G Limitations, failure cases and future work

A METHOD DETAILS

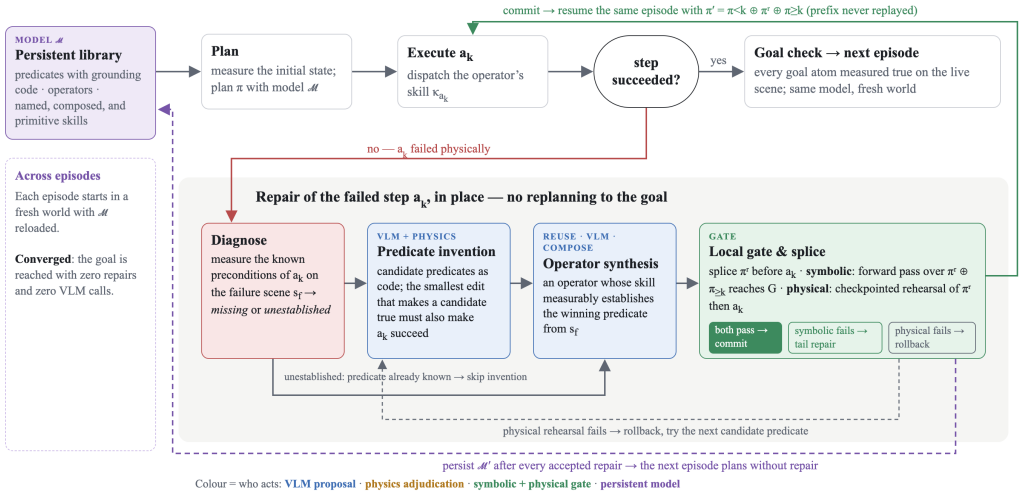


Figure 6: **Overview of SMOOTH.** Upon execution failure, our framework diagnoses the missing precondition, induces and physically verifies the required predicate and operator, splices the repair locally before the failed step, and persists the expanded domain for future tasks.

A.1 PSEUDOCODE

Algorithm 1 is one deployment episode, Algorithm 2 is predicate invention (§4.2), and Algorithm 3 is operator synthesis followed by the local gate (§4.3–§4.4). R is the number of retry rounds of rival proposals per repair, and B is the number of repairs per episode (default $B = 2$, overridable per task).

A.2 SCENE INTERFACE AND THE PREDICATE SANDBOX

Every predicate is a Python function `def check(scene, *args) -> bool` over a `Scene` object produced by ψ. The scene exposes `scene.xyz(name)` (position, m), `scene.poses[name]` (a 4×4 homogeneous transform), `scene.sizes[name]` (half-extents), `scene.axis_aligned_bbox(name)`, `scene.has_object(name)`, `scene.held` (the grasped object or `None`), and `scene.joint_states`, a dictionary of articulated joints *and* latent readings (e.g. `door_hinge`, `timer_dial`, `filled`). Code is compiled in a sandbox: `import`, dunder access, `open`, `exec`, `eval` and `compile` are rejected at parse time, builtins are restricted to a safe subset, and `numpy` is pre-bound as `np`. A predicate that raises on the failure scene is rejected with its exception as feedback. Predicates may also

Algorithm 1 One episode of SMOOTH

Require: task (ℓ, x_0, G, D), persisted model $\mathcal{M} = \langle \mathcal{P}, \mathcal{A} \rangle$, repair budget B

$\rho \leftarrow$ PYPERPLAN$(\mathrm{Abs}_{\mathcal{P}}(\psi(x_0)), G, \mathcal{A})$ ▷ initial plan; the only planner call
$k \leftarrow 1$; $n_{\text{rep}} \leftarrow 0$
while $k \leq |\rho|$ **do**
  $(x', b) \leftarrow \mathrm{Exec}(\kappa_{a_k}, x)$
  if $b = 1$ **then** $x \leftarrow x'$; $k \leftarrow k + 1$; **continue**
  end if
  if $n_{\text{rep}} = B$ **or** the same step failed with the same note after a committed repair **then**
    stop
  end if
  $s_f \leftarrow \psi(x)$; $F \leftarrow \{p \in \mathrm{pre}_{a_k} : \varphi_p(s_f) = 0\}$
  if $F = \emptyset$ **then** ▷ MISSING
    $p \leftarrow$ INVENTPREDICATE(a_k, x, D) ▷ Alg. 2; may return a set
    if $p = \bot$ **and** κ_{a_k} is parametric **then**
      retune the skill argument (§A.5)
    end if
    $\mathrm{pre}_{a_k} \leftarrow \mathrm{pre}_{a_k} \cup \{p\}$
  else ▷ UNESTABLISHED: $p \in F$ has no establishing operator
    $p \leftarrow$ the unestablished literal
  end if
  $(\rho^r, \mathcal{M}') \leftarrow$ SYNTHESIZEANDGATE(p, a_k, ρ, k, x) ▷ Alg. 3
  if $\rho^r = \bot$ **then** roll back pre_{a_k}; **stop**
  else $\rho \leftarrow \rho_{<k} \oplus \rho^r \oplus \rho_{\geq k}$; $\mathcal{M} \leftarrow \mathcal{M}'$; persist $\mathcal{M}$; $n_{\text{rep}} \leftarrow n_{\text{rep}} + 1$
  end if
end while
return GOALHOLDS$(G, \mathrm{Abs}_{\mathcal{P}}(\psi(x)))$

return a probability (`returns_probability`), which is the hook used by the soft programs of Appendix D.

The simulator exposes an *intervention space* $\mathcal{I}$ that is generic and scenario-agnostic: drive each articulated joint to each of its limits, set each latent reading to either end of its range, and remove each movable body. It never names which edit fixes a failure.

A.3 Failure Diagnosis

When a skill returns $b = 0$, SMOOTH re-measures every precondition of a_k on s_f. It distinguishes three cases. MISSING: all preconditions hold, so the model lacks one. UNESTABLISHED: a precondition is false and no operator in $\mathcal{A}$ adds it. STALE-EFFECT: a precondition is false although an earlier operator claims to add it, so that effect is too strong. MISSING runs the full pipeline. UNESTABLISHED names its target predicate and skips to operator synthesis. STALE-EFFECT is handled as MISSING, with a note in the prompt that an earlier effect may be over-claimed. The environment's own failure message is *never* shown to the VLM; the prompt states only that the step failed although every known precondition measured true.

A.4 Predicate Invention in Detail

Prompt contents. The proposer receives the instruction, the current plan and the failed step, the structured failure scene (poses, sizes, readings, held object), a rendered image (omitted for text-only backbones), the deployment context D, the *measured episode delta* $\Delta_{0 \to f}$ (which readings and poses changed between the initial and failure scenes), and the list of existing predicate names. It never sees the fix, the witness intervention, or the environment's failure note.

Rival hypotheses. The VLM is asked for 2–4 *competing* single explanations rather than one answer. Asking for rivals lets physics order them (below) rather than trusting the first plausible guess. With `atomic_predicates`, a rival whose code conjoins independent scene dimensions (e.g.

Algorithm 2 INVENTPREDICATE: rival hypotheses ranked by physics

Require: failed step a_k, failure state x_f, deployment context D

$L \leftarrow$ unattached library predicates that measure false on s_f and pass the semantic screen ▷ reuse first
$H \leftarrow \emptyset$ ▷ accumulated feedback
for $r = 1, \ldots, R$ **do**
  $C \leftarrow \mathcal{V}(\textsc{Prompt}(s_f, \text{image}, D, \rho, \Delta_{0\to f}, H))$ ▷ 2–4 rival programs φ_h
  drop $h \in C$ that do not compile, raise on s_f, or fuse independent dimensions (atomic split)
  for all $h \in L \cup C$ **do**
    if $\varphi_h(s_f) = 1$ **then** reject h (already true at failure); **continue**
    $E_h^\star \leftarrow \textsc{DirectedEdit}(\dim(\varphi_h), x_f)$ ▷ AST-read dimensions set to satisfying values
    if no $E \subseteq \mathcal{I}_h$, $|E| \le K$, makes φ_h true **then** try library skills as establishing witnesses
    $h.\textit{sufficient} \leftarrow \mathrm{Exec}(\kappa_{a_k}, x_{E_h^\star}) = (\cdot, 1)$ ▷ Eq. 1; from a checkpoint
  end for
  $S \leftarrow \{h : h.\textit{sufficient}\}$
  order S by asymmetric entailment ($h \succ h'$ if $E_h^\star$ also satisfies $\varphi_{h'}$; prefer the *weaker* h')
  break remaining ties by achievability with a named skill in $\mathcal{K}_\mathrm{N}$, then by position (library first)
  if $S \neq \emptyset$ **then return** the top of S
  end if
  if no candidate is sufficient alone **and** two or more were evaluated **then return** $\textsc{PredicateSet}(C)$ ▷ conjunction path
  end if
  $H \leftarrow H \cup \{\text{measured reason per rival}\}$; **if** $r \ge 2$ add the widen-search nudge
end for
return $\perp$

Algorithm 3 SYNTHESIZEANDGATE: establish p, splice, verify locally

Require: predicate p, failed step a_k, plan ρ, failure state x_f

for all candidate κ in order: library operators adding p (chains ≤ 3); VLM-chosen named skill in $\mathcal{K}_\mathrm{N}$; VLM-composed primitives $\langle \pi_1..\pi_m \rangle$ rolled out step by step **do**
  if not $\textsc{Establishes}(\kappa, p)$ from a checkpoint of x_f (Eq. 2) **then continue**
  $\mathrm{del}_a \leftarrow \{q : \varphi_q \text{ flips } 1 \to 0 \text{ under } \kappa\}$; $\mathrm{pre}_a \leftarrow$ discriminating predicates over perturbed starts (composed κ only)
  $\rho^r \leftarrow$ the smallest ordered segment of new operators after which a_k is applicable
  g2: walk STRIPS over $\rho^r \oplus \rho_{\ge k}$ from $\mathrm{Abs}_\mathcal{P}(s_f)$, threading effects; record first clobbered step m
  g3: from a checkpoint of x_f, execute ρ^r then a_k physically
  if g3 passes **and** g2 passes **then return** $(\rho^r, \mathcal{M} \cup \{a\})$ ▷ commit
  else if g3 passes **and** a library operator re-adds the literal clobbered at m **then**
    insert it before ρ_m and re-run g2 ▷ tail repair
  end if
  roll back a; feed the measured rehearsal lines back to the operator proposer
end for
return $\perp$

`door_closed_and_timer_set`) is dropped with feedback asking for atoms; a range on a single dimension is allowed. Each atom then maps to its own reusable operator.

Directed scene editing. Instead of sweeping $\mathcal{I}$, SMOOTH parses the candidate's AST to find the dimensions $\dim(\varphi_h)$ it reads (joint keys, object names) and constructs the edit that sets exactly those dimensions to values satisfying the predicate, up to K simultaneous edits. The edited state is settled by the simulator before the failed skill is re-run from the checkpoint. If no edit in $\mathcal{I}$ can make φ_h true (an *out-of-span* condition such as a kettle's fill level), each library skill is tried as an establishing witness (`validator_use_skills`), with bounded sequences optional.

Entailment ranking. Several rivals can be sufficient at once. A proxy ("the tray was moved") and a cause ("the cup was moved") may both let the skill succeed. For sufficient h, h', we test whether the minimal edit $E_h^\star$ also satisfies $\varphi_{h'}$. If it does and not vice versa, h' is the weaker, more general condition and is preferred. Remaining ties are broken by whether a named skill can bring the state about (*achievability*), which separates an actionable cause from an unreachable confound. After that, library candidates win, which avoids duplicating a known predicate.

Feedback and retries. Each rejected rival contributes one line of *measured* feedback to the next round, e.g. "`timer_set`: the smallest way to satisfy it is 1 intervention, but under that `press_start` still does NOT succeed". Feedback accumulates across rounds, because showing only the latest lesson let models revert to accessors they had already been told do not exist. After two rejected rounds a generic *widen-search* nudge is appended: readings not yet read, what each skill changes, several conditions at once, and the model's own prior repairs. If no rival is sufficient alone but several were evaluated, the loop falls back to the predicate-set path, where the VLM proposes a conjunction that is tested as a single hypothesis and repaired as a multi-step fix.

Library reuse. Before any VLM call, unattached library predicates that are false on s_f are screened physically, then semantically (a short VLM verdict on whether each applies). The survivors join every round's rival list, listed first.

A.5 Operator Synthesis in Detail

Operators are searched in order of increasing risk: an existing operator that adds p (or a chain of up to 3); a named skill the VLM maps to p from the skill list alone; and a composition of primitives. For compositions, the VLM proposes a primitive sequence and each primitive is executed from the checkpoint in turn. The measured state after each step is returned, so a broken sequence is repaired from the step where it went wrong rather than regenerated open-loop. Delete effects are fitted from physics: every known predicate that flips $1 \to 0$ when κ runs from x_f is added to del_a. For composed skills, which have no authored preconditions, x_f is perturbed and the smallest predicate set that separates successful from failed rollouts becomes pre_a (`learn_composed_preconditions`). When the failed step's skill is *parametric* and no predicate explains the failure, the VLM proposes new continuous values (e.g. an approach angle), which are rehearsed exactly like an operator.

A.6 The Local Gate in Detail

The repair segment ρ^r is ordered so that each new operator's preconditions hold when it is reached, then trimmed to the shortest prefix after which a_k becomes applicable (`trim_repair_segment`). **g2** is a single linear STRIPS pass over $\rho^r \oplus a_k \oplus \rho_{>k}$ from the *measured* failure state, threading add and delete effects. It therefore catches a repair that clobbers a literal an intermediate step a_m needs, even when the goal atoms are untouched, and it credits later steps that legitimately re-establish a clobbered fact. No search runs. **g3** restores a checkpoint of x_f and executes only ρ^r and then a_k. The already-executed prefix is not replayed, because the live state already reflects it. On a g2 failure with a g3 pass, *tail repair* inserts a library operator that re-adds the clobbered literal immediately before a_m, preferring one whose preconditions already hold there. Any other failure rolls back every addition of the repair: predicates, attachments and operators.

A.7 Persistence and Reuse

After a commit, the predicate library (code, description, argument types, provenance), the operator library (STRIPS schema, skill binding, composition recipe) and the precondition attachments are written to one JSON file. At the next episode, attachments whose operator no longer exists are quarantined, attachments are welded into the operator schemas they belong to, and composed skills are re-registered with the simulator. Convergence is the first episode that plans and executes to G with zero repairs and zero VLM calls. Across all converged nightly episodes (Appendix C), every turn after convergence made exactly zero VLM calls.

A.8 Ablation Flags

Every algorithmic component is behind a flag, and Table 7 lists the ones used in this paper. Defaults are the configuration evaluated in §5. The nightly sweeps (§5.5) compare `baseline` (scenario defaults) against `all` (`atomic_predicates` and `broad_reuse` forced on).

Table 7: **Ablation flags** (subset) and their defaults.

Flag	Default	Effect
`local_gate`	on	accept a repair by splicing and linear suffix validation (g2) plus delta rehearsal (g3); off re-plans from the failure state
`entailment_ranking`	on	ask for rival hypotheses; keep the sufficient ones; prefer the weaker under asymmetric entailment
`directed_scene_edit`	on	construct the satisfying edit from the dimensions the predicate reads (AST) instead of sweeping $\mathcal{I}$
`validator_focus`	on	restrict interventions to the predicate's dimensions
`validator_max_compound`	1	max simultaneous edits K (scenario-raised for conjunctive failures)
`validator_use_skills`	off	use the agent's own skills as witnesses for out-of-span conditions
`allow_predicate_set`	on	allow a conjunction of predicates as one hypothesis (multi-step fix)
`atomic_predicates`	on	reject rivals that fuse independent dimensions
`broad_reuse`	on	reuse a library operator that already adds the predicate before inventing
`skill_composition`	on	compose primitives when no named skill establishes the predicate
`learn_composed_preconditions`	on	learn a composed skill's preconditions from perturbed rollouts
`tail_repair`	on	patch a clobbered downstream literal with a known operator
`trim_repair_segment`	off	cut the segment at the shortest applicable prefix
`parametric_skills`	off	allow continuous skill-argument retuning
`episode_delta_in_prompt`	on	include the measured initial-to-failure delta in every proposal prompt
`skill_effect_feedback`	on	report each skill's measured effects when a candidate is ruled unachievable
`retry_nudge` / `nudge_after`	on / 2	widen the search after 2 rejected rounds

The physical-rehearsal ablation (Table 6) is a separate harness arm, not a registry flag. It disables the hypothesis test, entailment ranking's physical filter, the establishment probe and g3 together, and keeps g2.

B Prompts

The prompts are generic across scenarios; nothing names a failure mode or a fix. Listing 1 is the rival-hypothesis system prompt, which is the one ablated in Appendix C.1. Its three guidance blocks are marked [POLARITY], [CONTRACT] and [REASONING]. The per-failure user prompt opens with the text in Listing 2, followed by the structured scene, the plan, the deployment notes and the measured episode delta.

Listing 1: Rival-hypothesis system prompt (verbatim; em-dashes rendered as $--$).

```
You repair a robot's symbolic model after a skill failed. A step failed even though every KNOWN precondition
      held,
so the model is MISSING a precondition. Propose SEVERAL RIVAL HYPOTHESES for what that missing precondition
      was --
competing explanations, NOT a conjunction. Each one alone should be a complete candidate answer.

[polarity]  WRITE THE REQUIREMENT, NOT THE FAULT
A precondition is a condition that must HOLD for the step to work. So it must measure FALSE at the moment the
      step
failed, and TRUE once the world is put right. Write `door_is_open` (true when the door stands open), never
`door_not_open_enough` (true right now, while the step is failing). A predicate that already holds at the
      failure is
discarded immediately -- it cannot be what was missing. If a name of yours contains "not", "too" or "un-", you
       have
almost certainly written the fault; turn it around.

[contract]  HOW THEY ARE JUDGED -- read this first
Your predicates are COMPILED AND EXECUTED. Each is tested separately: the world is edited until that predicate
measures True, and the failed skill is re-run. Candidates that do not make the skill succeed are discarded.
      Among
those that DO, the robot then compares them: if satisfying one of your predicates also makes another one true,
       the
first is the stronger claim and the SECOND -- the more general condition -- is preferred. So you are not
      penalised
for offering a guess that turns out to be a side effect; offer the obvious reading AND the underlying one and
      let
physics order them.

Aim for 2-4 genuinely different explanations, each reading DIFFERENT scene state whenever you can.
Return STRICT JSON:
  {"reasoning": "<your working -- WRITE THIS FIELD FIRST>",
   "predicates": [ {"name": "<snake_case>", "args_types": [], "intended_args": [],
```

```
                    "description": "<what scene state it reads>",
                    "code": "def check(scene):\n    ...\n    return <bool>"}, ... two to four rival candidates
      ...]}
Each predicate reads scene.joint_states / scene.xyz(name) / scene.has_object(name) / scene.held.

ARGUMENTS. Prefer `args_types: []` -- a predicate about fixed scene furniture needs none. If a predicate IS
     about
particular objects, list their KINDS in `args_types` AND the actual scene objects you mean in `intended_args`,
      ...

[reasoning]  THINK BEFORE YOU ANSWER. Begin your JSON object with a "reasoning" field and write it FIRST: what
      the
numbers say went wrong, which objects the property is about, and what each predicate would measure on the
     failure
scene versus once the world is put right. It is a scratchpad -- not executed, not scored.
```

Listing 2: Opening of the per-failure user prompt and the operator-proposer task.

```
PREDICATE:  the step {step} returned failure, yet every known precondition of it still measures True on the
     scene
            below. Some precondition is therefore MISSING from the model. Read the scene state (object poses,
     joint
            readings, gripper) and the image and hypothesise which single condition, had it held, would have
     let
            the step succeed.
OPERATOR:   The invented predicate is the missing precondition of the failed step. Choose the skill(s) from
     the
            library whose execution ESTABLISHES it (makes it measure true). Reason from the predicate's
     meaning and
            the skill names; your choice is checked by a physical rehearsal, and you are re-asked if it does
     not
            establish the predicate.
RETRY (v1): WHAT YOU HAVE ALREADY TRIED -- every lesson here still applies: <one measured line per rejected
     rival>
            Propose DIFFERENT hypotheses. Do not repeat any predicate above, ...
```

C Additional Ablations

Unless stated otherwise, the numbers below come from the ten nightly sweeps described in Appendix F (Table 8): 1,366 episodes over 8 anchors, 3 seeds, 3 GPT backbones and 2 flag configurations, at temperature 0.7. They reuse the logged events; no episode was re-run for this section except the prompt ablation and the predicate-observation study, which use Sarvam-105B (Sarvam AI, 2026) and Jev (TypeSafe AI, 2026) on a small subsample.

Table 8: **Nightly sweeps** used for the ablations. Converged = sweep success criterion (Appendix F). [a]Model recorded as `openai`; the workflow pinned gpt-4o. [b]Cancelled mid-sweep; completed episodes are used, and configurations with < 3 seeds are not eligible for best-configuration selection.

Sweep	Date	Models	Flag-sets	Episodes	Converged
1	09-20	gpt-4o[a]	baseline, all, atomic, broad_reuse, op_precond	120	55
2[b]	09-21	gpt-4o, gpt-4o-mini	baseline, all	94	34
3–6	09-21 – 09-23	gpt-4.1, gpt-4o, gpt-4o-mini	baseline, all	4×144	82, 77, 81, 80
7–10	09-23 – 09-25	gpt-4.1, gpt-4o, gpt-4o-mini	baseline, all	4×144	83, 77, 79, 76
Total				**1,366**	**724**

C.1 Prompt Engineering

Setup. We ablate the rival-hypothesis prompt (Listing 1) one block at a time. `no-polarity` removes "write the requirement, not the fault". `no-contract` removes the description of how hypotheses are judged. `no-reasoning` removes the reasoning-first scratchpad, and its JSON field. `no-feedback` keeps the retry rounds but replaces the measured per-rival lessons with "your previous hypotheses were all rejected". `minimal` removes all four. Everything else, including the physical tests, is unchanged. To use a different model family from the nightly GPT runs and to keep cost low, the ablation runs Sarvam-105B (Sarvam AI, 2026), a text-only model, so the image is withheld and the model reasons from the structured scene alone. It covers 4 anchors (microwave, outofspan, confound, relational), 1 seed, default flags, temperature 0.2, and 2 turns per episode.

Table 9: **Prompt ablation** (Sarvam-105B, text-only, 1 seed). Per-scenario success (✓/✗); totals over the 4 episodes. Calls = mean VLM calls per episode; Trunc. = predicate-proposal replies lost to a truncated (unparseable) JSON reply at the 4,096-token cap; Rej. = hypotheses rejected by the validator / tested.

Prompt	micro.	outofspan	confound	relat.	Succ.	Calls	Trunc.	Rej.
full prompt (v1)	✓	✗	✗	✗	1/4	11.0	31/38	1/2
− polarity	✗	✓	✗	✗	1/4	10.0	30/37	1/2
− contract	✓	✗	✗	✗	1/4	10.8	29/36	0/2
− reasoning	✓	✓	✓	✗	3/4	8.0	10/18	1/4
− retry feedback	✗	✗	✗	✗	0/4	12.5	40/45	0/1
minimal (all four removed)	✓	✓	✓	✗	3/4	9.2	12/22	0/3

Observations. (i) *The prompt that works for GPT fails on Sarvam, and the reason is the output budget, not the reasoning.* With every variant that keeps the reasoning block, 80–89% of Sarvam's predicate-proposal replies never reach the validator. The model fills the JSON `"reasoning"` field until it hits the 4,096-token cap, and the truncated reply does not parse. A logged rerun of *outofspan* under v1 confirmed that each lost reply was an unterminated {`"reasoning": ...` string. Removing the block cuts the loss to 55% and lifts success from $1/4$ to $3/4$ in fewer calls (8.0 vs. 11.0). The block was tuned on gpt-4.1. A reasoning scaffold is therefore a per-backbone choice. With a verbose model it should be capped, moved to the provider's native reasoning channel, or dropped. (ii) *Removing retry feedback is the only change that does worse than v1.* Replacing the measured per-rival lessons with a generic "all rejected" gives $0/4$, the worst row, and has the most calls (12.5) and the most truncation ($40/45$). The gap to v1 is a single episode, so we read it only as consistent with the nightly finding that most recovered episodes are recovered by the retry loop (Section C.2). (iii) *Polarity and contract are not resolved here.* So few replies parsed that 1–4 hypotheses per variant reached the validator, so this run cannot show their effect. The nightly logs are the better evidence for polarity. There, polarity is the second-largest rejection reason, and it grows as the backbone weakens (Table 10). (iv) *Relational fails under every prompt.* Its user prompt is the longest (five objects and their relations), and even without the reasoning block 9 of 12 replies overran the cap. With $n = 1$ per cell, these rows are qualitative. The most solid signal is the truncation rate. It falls from 82% to 56% when the reasoning block is removed, and the drop shows up in each of the three scenarios where truncation occurs (microwave never truncated).

C.2 Why VLM-Proposed Predicates Are Rejected

Table 10 classifies every rejected hypothesis test in the nightly logs by the measured reason the validator returned. It also counts rivals dropped before testing (fused conjunctions) and episodes that crashed.

Table 10: **Rejected predicate hypotheses by reason** (share of each backbone's rejections, all ten sweeps), plus rivals dropped before testing and crashed episodes.

Reason	gpt-4.1	gpt-4o	gpt-4o-mini
World cannot make it true (unreachable)	48.3%	64.9%	47.0%
Already true at the failure (wrong polarity / wrong object)	28.0%	22.6%	35.1%
Satisfiable, but the skill still fails (insufficient)	12.2%	10.1%	4.4%
Code raises on the scene	10.6%	2.2%	8.0%
Skill succeeds without it (not necessary)	0.9%	0.1%	5.5%
Rejected tests (total)	847	989	1,470
Fused conjunctions dropped before testing	1	6	43
Crashed episodes: forbidden token (`import`) / syntax / API	35 / 0 / 0	4 / 7 / 6	3 / 23 / 1

Observations. (i) *Most rejections are not wrong ideas but unreachable ones.* Almost half of all rejections are predicates that read a quantity no intervention can move: a sensor the scene does not have, an invented accessor, or a threshold outside a joint's range. These are exactly the claims a text-only critic cannot refute and a physical test refutes immediately. (ii) *Polarity errors are the second failure mode*, and the weakest model makes the most of them (35.1%). The model describes the fault (*shelf_approach_angle_valid* is already true) instead of the requirement. This is the error the polarity

block of the prompt targets (Appendix C.1). (iii) *Each backbone has a characteristic code failure.* gpt-4.1 writes the most defensive code, calling `hasattr` and importing `math`; this accounts for all 35 of its crashes and most of its 10.6% runtime rejections, even though its hypotheses are the most often correct. gpt-4o-mini emits syntax errors (23 crashes) and fused conjunctions (43 drops). A sandbox that exposes a slightly larger, documented builtin set would remove most of gpt-4.1's crashes without weakening safety. (iv) Rejections are rarely fatal but are costly. Among episodes with at least one rejected test, 15% (gpt-4.1), 27% (gpt-4o) and 26% (gpt-4o-mini) still converged, against pooled success of 59/58/42%. A rejected first guess is thus a strong predictor of a hard repair, although the retry loop does recover a sizeable fraction.

C.3 Flag Configurations

Table 11 pairs every nightly episode run under `baseline` with the same (sweep, model, scenario, seed) under `all` (646 pairs).

Table 11: **Flag configurations.** Left: paired `baseline` vs. `all` (646 pairs) per scenario (successes). Right: the five flag-sets of the first sweep (gpt-4o, 24 episodes each).

Scenario	Pairs	baseline	all
confound	81	59	57
largespace	81	44	44
microwave	81	78	78
obstacle	81	13	**18**
outofspan	81	**58**	52
parametric	79	24	24
relational	81	25	**40**
transfer	81	38	42
Total	646	52.5%	**55.0%**
VLM calls / ep.		7.57	**6.86**
Crashed		**35**	44

Flag-set	Succ.	Calls
baseline	13/24	8.42
atomic	10/24	8.46
broad_reuse	10/24	9.00
op_precond	10/24	8.38
all	12/24	**7.50**

Observations. The effect is scenario-specific, not uniform. Forcing atomic predicates and broad reuse helps the two scenarios whose fix is a spatial relation over several objects: *relational* ($25 \rightarrow 40$) and *obstacle* ($13 \rightarrow 18$). There a fused relational predicate is hard to establish with any single operator. It slightly hurts *outofspan* ($58 \rightarrow 52$), where the fix is a single latent dimension and decomposition has nothing to split. Across all pairs `all` wins 87 episodes that `baseline` loses and loses 71 that `baseline` wins, while using 9% fewer VLM calls. In the first sweep, no single flag beat the baseline alone. Their combination did, and it used the fewest calls. The flags interact: atomic predicates generate more reusable atoms, and broad reuse is what finds them.

C.4 Repair vs. Replan, Extended

Replan arm. For each nightly episode, the scenario is replayed on the source code of the commit that produced it. The learned library is cut back to its state at the end of each turn, using the library snapshot that opens the next turn, so that turn-t replanning uses only turn-t knowledge. Handing Replan the final library instead leaks later-turn knowledge and inflates its success (812 vs. 724 converged). Seeded operators absent from a snapshot are kept, and invented ones absent from it are dropped. 39 episodes whose learned operators were quarantined in a later turn cannot be reconstructed and are excluded. 28 further episodes whose libraries contain uncompilable code count as Replan failures; SMOOTH also failed all 28. As a consistency check, Replan's initial plan matched SMOOTH's logged plan on every best-configuration episode.

The seven turn-1 cases only SMOOTH solved. In every one of them the learned symbolic model was imperfect, and replanning trusted it:

- *Over-claimed effect* (microwave, $\times 2$). The learned `ensure_timer_set` declared the fused effect *door_closed_and_timer_set* although its skill only sets the timer. Pyperplan took the cheapest route, `ensure_timer_set` $\rightarrow$ `press_start`, skipped closing the door, and the microwave did not start. SMOOTH had rehearsed both operators in sequence and kept both.

Table 12: **Repair vs. replan over all** 1,327 **comparable episodes.** Left: per scenario, successes and mean D_{sym} on turn-1 successes. Right: paired outcomes (same episode, same knowledge).

	Success		D_{sym}	
Scenario	SMOOTH	Replan	SMOOTH	Replan
confound	124	124	1.44	1.00
largespace	90	90	2.01	2.01
microwave	163	163	2.00	2.00
obstacle	31	**32**	**1.24**	1.75
outofspan	116	116	1.00	1.00
parametric	48	48	0.00	0.00
relational	65	65	1.67	1.01
transfer	**86**	84	1.22	1.00

	Both	SMOOTH only	Replan only	**Neither**
Turn-1 goal	733	**7**	0	587
Success	721	**2**	1	603

- *Mis-scoped precondition* (transfer, $\times 2$). The attached *cabinet_door_angle_correct* was guarded on `is_cabinet_door ?x`, so it did not constrain grasping the mug, and the planner legitimately dropped `open_cabinet`.
- *Meaningless binding of a widened precondition* (obstacle, transfer). For example, `place_widget_on_pad(widget, pad, widget, widget)` satisfied the symbolic model without the obstacle ever being moved.
- *A learned operator that does not physically clear the path* (relational). The planner chose it, whereas SMOOTH's rehearsed segment used the one that does.

Physical rehearsal thus protects the executing plan from a class of model errors that symbolic replanning cannot detect: the segment SMOOTH splices is the one that was *observed* to work, whatever the symbolic model says about alternatives. The converse effect is visible in the pooled D_{sym}, where SMOOTH is higher on confound, relational and transfer. At those commits the repair segment instantiated *every* ground instance of an invented operator (e.g. `cabinet_door_open_via_open_cabinet` for the mug, the kettle and the plate) where one suffices. The `trim_repair_segment` flag removes these, and on its own it would bring SMOOTH's pooled D_{sym} to or below Replan's.

C.5 VLM Backbone, Extended

Table 13: **VLM backbone.** Best configuration per scenario (8 scenarios $\times$ 3 seeds); the last column pools all runs and configurations. Predicate rejection is the fraction of VLM-proposed predicates refuted by physical hypothesis testing; operator rejection is the fraction of proposed repairs rolled back by the local gate.

Backbone	**Succ.**	**Pred. rej.**	**Op. rej.**	**Calls↓**	**Pooled succ.**
gpt-4.1	**87.5%**	30.3%	2.4%	3.58	**58.6%**
gpt-4o	83.3%	24.1%	2.9%	**3.25**	57.6%
gpt-4o-mini	70.8%	66.0%	0.0%	3.71	42.1%

Best configuration. Table 13 shows that SMOOTH degrades gracefully with a weaker VLM. gpt-4.1 and gpt-4o are within 4.2 points of each other, and gpt-4o-mini still solves 70.8%. Scenario difficulty is backbone-specific: only gpt-4.1 solves *parametric* ($48/48$ pooled vs. $0/67$ for gpt-4o), while it is weakest on *transfer*. The rejection rates show where the weaker model's errors are caught. gpt-4o-mini's predicate rejection is more than twice that of the larger models (66.0% vs. 24–30%), so physical hypothesis testing filters its incorrect proposals before they reach the domain. Operator rejection stays below 3% for all backbones: once a predicate survives testing, the establishing operator is almost always correct. The pooled numbers preserve the ranking (58.6/57.6/42.1%) with per-sweep standard deviations under 4 points for gpt-4.1 and gpt-4o-mini.

gpt-4o's minimum is the cancelled partial sweep; excluding it, gpt-4o ranges over 45.8–72.9%. The weakest backbone has the highest predicate rejection and needs the most calls. Its low operator rejection reflects that fewer of its proposals ever reach the gate, not that its operators are better. Sce-

Table 14: **VLM backbone over all sweeps and configurations** (no selection). Per-sweep success is mean ± s.d. (min–max) over sweeps.

Backbone	Ep.	Success	Per-sweep	Pred. rej.	Op. rej.	Calls	Crashed
gpt-4.1	384	**58.6%**	58.6 ± 3.8 (52.1–64.6)	74.0%	13.6%	**5.52**	35
gpt-4o	550	57.6%	59.3 ± 13.7 (26.1–72.9)	**67.9%**	19.0%	6.98	17
gpt-4o-mini	432	42.1%	42.1 ± 3.7 (37.5–47.9)	81.0%	**6.8%**	9.22	27

nario difficulty is strongly backbone-specific: gpt-4.1 solves *parametric* in $48/48$ pooled episodes while gpt-4o and gpt-4o-mini solve $0/67$ and $0/54$, yet gpt-4.1 is the weakest on *transfer* ($5/48$).

D Evaluating Learned Predicates on an Imperfect Twin or a Real Robot

Motivation. SMOOTH learns predicates as crisp code over the simulator's scene, and that code is exact on the twin by construction. On a real robot, or a twin that only approximates it, two things change. Poses are noisy, and many *non-geometric* readings (a timer dial's angle, a kettle's fill level, a latch) are not observable at all. We ask how the learned predicates should be *evaluated* in that regime, and compare SMOOTH's crisp code with two recent alternatives. The first is a calibrated text judge, Jev (TypeSafe AI, 2026), a "System One" model that returns a probability for a yes/no question over structured state. The second is a NePTune-style soft program (Kamali & Kordjamshidi, 2026): NePTune translates a query into Python that mixes imperative control flow with fuzzy-logic operators over model-scored atomic concepts.

Data. We replay turn 1 of two converged episodes per anchor (sweeps 9–10, the same commit): the seed plan until it fails, then the plan from the episode's end-of-turn-1 library. We keep the scene after every step together with the executed-action history. Each learned predicate is evaluated with its own code on the perfect twin to give the label, for groundings whose value changes over the episode. This yields 18 distinct learned predicates (12 non-geometric, reading `joint_states`; 6 geometric) and 136 (predicate, grounding, scene) triples, 80 of them positive.

Observation conditions. *Twin*: the exact scene. *Noisy twin*: positions $+\mathcal{N}(0, 1\,\text{cm})$ and readings $+\mathcal{N}(0, 0.05)$. *Real robot (blind)*: noisy positions, all joint and latent readings withheld, and the executed-action history (e.g. `fill_kettle(kettle): ok`), which a real robot always has.

Classifiers.

- **Crisp**: the learned code on the observation; an exception counts as false.
- **Jev**: one Noul per predicate, with the predicate's name, arguments and learned description as the question and the compact observation plus history as state. It does not see the code. One request per scene batches all its predicates.
- **NePTune-style**: Sarvam-105B rewrites each learned predicate once, from its code and description, as `check(scene, score, *args)`, which returns a probability. The program uses sigmoid comparisons (`soft_lt`/`soft_gt`), $\min / \max / 1-p$ connectives, and `score(q)` (a Jev Noul over the observation and history) for any condition it cannot compute from the observation. An example is in Listing 3.
- **Crash fallback**: crisp, or Jev when the code raises.
- **Reading-aware routing**: crisp, or Jev when the predicate reads a reading the observation withholds. This is known statically, because SMOOTH already extracts $\dim(\varphi_p)$ from the AST for directed editing.

Listing 3: NePTune-style rewrite of the learned *timer_set* (generated by Sarvam-105B, verbatim). The learned crisp version is `return scene.joint_states['timer_dial'] > 0.05`.

```
def check(scene, score, *args):
    dial = None
    if scene.joint_states and 'timer_dial' in scene.joint_states:
        dial = scene.joint_states['timer_dial']
```

Table 15: **Evaluating learned predicates under imperfect observation.** Accuracy at threshold 0.5 (Brier score in parentheses) against the perfect-twin label; 136 triples (90 non-geometric, 46 geometric). 342 Jev requests and 18 Sarvam generations in total.

	Twin			Noisy twin			Real robot (blind)		
Classifier	all	non-geo	geo	all	non-geo	geo	all	non-geo	geo
Crisp (SMOOTH)	**1.00** (0.00)	**1.00**	**1.00**	**.926** (.074)	**.900**	**.978**	.610 (.390)	.422	**.978**
Jev	.765 (.134)	**1.00**	.304	.691 (.186)	.889	.304	.743 (.163)	.944	.348
NePTune-style (Sarvam + Jev)	.787 (.157)	.844	.674	.809 (.170)	.878	.674	.882 (.155)	**.989**	.674
Crash fallback	**1.00** (0.00)	**1.00**	**1.00**	**.926** (.074)	**.900**	**.978**	.816 (.206)	.733	**.978**
Reading-aware routing	**1.00** (0.00)	**1.00**	**1.00**	**.926** (.074)	**.900**	**.978**	**.956** (**.050**)	.944	**.978**

```
    else:
        return score("Is the microwave timer dial set to a positive value?")
    return soft_gt(dial, 0.0, 0.02)
```

Observations.

1. *Learned code is the right representation for geometry, at every noise level.* Crisp code keeps 97.8% accuracy on geometric predicates under noise and in the blind condition. Jev, asked the same question in words, is *anti-correlated* with the label (30–35%). It answers "is the path to the pad clear" by plausibility over the action history ("the obstacle was moved, so it is clear") rather than by the interval arithmetic the question requires. A System One judge is not a geometry engine, which supports SMOOTH's choice to invent executable predicates rather than ask a model.
2. *Non-geometric state is recoverable from what the robot did.* When the readings are withheld, crisp code collapses to chance on non-geometric predicates (42.2%, balanced accuracy 0.50). Jev recovers 94.4% from the action history alone, with good calibration (Brier 0.064), because "`turn_timer(microwave)`: ok" is strong evidence for *timer_set*. The symbolic model can supply this evidence directly, since it records which operators ran.
3. *Crisp code fails silently, not loudly.* Of the 90 blind non-geometric evaluations, 49 raised, and the crash fallback catches those. But 24 of the 41 that did *not* raise were wrong: code such as `scene.joint_states.get('lid_latch', 0.0) >= 1.0` quietly substitutes a default for a missing sensor. The crash fallback therefore reaches only 73.3% on non-geometric predicates. Routing on the *dimensions a predicate reads*, which SMOOTH already computes for directed editing, catches both kinds and gives the best overall result (95.6%, Brier 0.050).
4. *NePTune-style soft programs are the best single representation off the twin, and the worst on it.* The Sarvam-written programs reach 98.9% on non-geometric predicates in the blind condition, because each program learned *when* to defer to `score`. They are the best single classifier there (88.2% overall). On the perfect twin, however, they lose accuracy (84.4% non-geometric), because rewriting `x > 0.05` as `soft_gt(x, 0.0, 0.02)` moves the decision boundary: a dial at exactly 0 scores 0.5. One of the six geometric programs referenced an undefined helper (`NameError` on all 18 of its triples); excluding it, NePTune-style geometric accuracy is 89.3%. Soft relaxation helps under noise only if its temperature and centre are fitted to the sensor, not guessed by the LLM.

SMOOTH vs. NePTune and Jev. NePTune composes pre-trained concept scorers into programs for *visual question answering*. Its programs are generated per query, are not validated against physics, and are not persisted. SMOOTH invents the *concepts themselves* from execution failures, validates each against physical interventions, and stores them as reusable planning symbols. Jev provides calibrated atomic judgments but no mechanism for deciding which judgment a plan needs. The results suggest that the approaches compose well. SMOOTH decides *which* predicate is missing and verifies it on the twin. Crisp code evaluates it wherever its dimensions are observed. A calibrated judge, or a NePTune-style soft program that calls one, evaluates it where they are not, with the robot's own action history as evidence.

Future work. (i) Fit soft thresholds, i.e. each sigmoid's centre and temperature, from the twin's measured noise instead of letting the LLM guess them. The directed-editing machinery already

produces labelled boundary cases. (ii) Use the calibrated probability as a first-class signal in the gate: an uncertain precondition on the real robot could trigger a sensing or confirmation action before a skill runs, rather than a silent default. (iii) Validate predicates *on the real robot* through the same hypothesis test, with the real execution outcome as the label and the twin supplying only candidate edits. (iv) Extend the static routing to image-based judges for non-geometric states with a visual signature (steam, an indicator light), which a text-only judge cannot see.

E Learned Abstractions

Listings 4–5 show predicates learned by SMOOTH (gpt-4.1; code verbatim, comments shortened, the relational listing reflowed) and the operators synthesized to establish them. They are representative of what the library holds after convergence: short, readable, and reading exactly the state the failed skill needed.

Listing 4: Learned predicates for the microwave (conjunctive, Group 3), the out-of-span kettle (Group 2) and the autoclave confound (Group 2).

```
# microwave: "the microwave timer dial is set to a nonzero value"      established by:
    set_timer
def check(scene):
    return scene.joint_states['timer_dial'] > 0.05
# microwave: "door closed if hinge is nearly 0"                        established by:
    close_door
def check(scene):
    return abs(scene.joint_states['door_hinge']) < 0.05
# outofspan: "the kettle contains liquid"   (no generic edit reaches it; validated via the
    skill fill_kettle)
def check(scene):
    return scene.joint_states.get('filled', 0) > 0
# confound: "the autoclave lid is fully latched"   established by: close_lid (ranked over
    indicator_lit)
def check(scene):
    return scene.joint_states.get('lid_latch', 0.0) >= 1.0
```

Listing 5: Learned relational predicate (relational, Group 2): a quantified condition over every other object, which transfers across a layout swap.

```
# relational: "no object other than the widget overlaps the pad's (x, y) area"   established
    by: clear_pad
def check(scene):
    px, py, _ = scene.xyz('pad'); w, l, _ = scene.sizes['pad']
    for name in scene.poses:
        if name == 'pad':
            continue
        ox, oy, _ = scene.xyz(name); ow, ol, _ = scene.sizes[name]
        if (ox + ow/2 > px - w/2 and ox - ow/2 < px + w/2 and
                oy + ol/2 > py - l/2 and oy - ol/2 < py + l/2):
            return False
    return True
```

The persisted microwave library adds two invented operators and attaches both predicates to the seed step (gpt-4.1, verbatim names). The delete effect `(not (door_open))` was fitted from physics, not proposed:

```
(:action door_closed_via_close_door  :parameters ()  :precondition (and)
   :effect (and (door_closed) (not (door_open))))                ; skill: close_door
(:action timer_set_via_set_timer     :parameters ()  :precondition (and)
   :effect (and (timer_set)))                                    ; skill: set_timer
; attachments: press_start  +=  (door_closed) (timer_set)
```

F Benchmark and Evaluation Protocol

Anchor scenarios. The ablations use eight OpenRepairBench anchors, each with one designed failure and its own persona (a README listing objects, observable joints and skills, never failure modes). Tags give the bench's structure / fix locus / identifiability axes.

- *microwave* (conjunctive / in-span / clean, kitchen): `press_start` needs the door closed *and* the timer set, which is a 2-edit witness.

- *confound* (atomic / in-span / confound, clinic): an autoclave whose indicator lamp is coupled to the lid interlock. The rivals *indicator_lit* and *lid_latched* are both plausible, and directed editing finds the independent lamp-test route, so only *lid_latched* ranks as sufficient.
- *relational* (relational / in-span / clean, desk): placement needs the pad clear, via a landmark-typed argument that must transfer across a layout swap.
- *outofspan* (atomic / out-of-span / clean, kitchen): pouring needs a filled kettle, a latent reading no generic edit reaches, only the skill `fill_kettle`.
- *largespace* (atomic / in-span / distractor, desk): many controls, few relevant; hypothesis-directed focus cuts the search cost.
- *transfer* (atomic / reuse / clean, kitchen): turn 1 (fetch) learns `open_cabinet`; turn 2 is a *different* task (stow) that reuses it; turn 3 converges.
- *obstacle* (negative / in-span / clean, desk): an obstacle fouls the approach corridor; the repair invents a path-clear predicate and a sweep operator, and widens the placement operator.
- *parametric* (atomic / parametric / clean): there is no missing precondition; the fix is a continuous skill parameter (the approach angle).

Ablation protocol. Unless stated otherwise, the ablations of Section 5.5 use the eight anchors above with 3 seeds each. They draw on ten independent nightly sweeps (1,366 episodes; Table 8) over three GPT backbones (gpt-4.1, gpt-4o, gpt-4o-mini) and two flag configurations (the default, and atomic predicates with broad library reuse), at temperature 0.7. For each scenario we report the best configuration over its 3 seeds (selection rule below) alongside the pooled numbers over all runs.

Metrics. *Success*: every turn reaches G and the episode converges (a turn with zero repairs; for *transfer*, a turn ≥ 2). Crashed episodes count as failures. D_{sym} (Fox et al., 2006) is computed over multisets of ground actions between the turn-1 plan ρ and the steps executed successfully in turn 1, on turn-1 successes only. *Predicate rejection* is the fraction of hypothesis tests of VLM-proposed predicates, single or compound, that rejected the proposal. *Operator rejection* is the fraction of local-gate verdicts on a proposed repair that rolled it back. Crashes are counted separately, not as rejections.

Best-configuration selection. For each scenario and backbone, the "best configuration" is the (sweep, flag-set) configuration with the highest success over its 3 seeds, breaking ties by fewer mean VLM calls and then by the newer sweep. This takes a maximum over 56–57 configurations per scenario, so it is optimistic. Tables 12 and 14 report the unselected pooled numbers alongside.

G Limitations, Failure Cases and Future Work

- **Rehearsal needs a checkpointable twin.** All physical tests restore simulator checkpoints. Rehearsal and establishment probes are *skipped while the gripper holds an object*, because restoring a checkpoint does not preserve a grasp. The repair is then admitted on the symbolic check alone. Appendix D is a first step toward evaluating the learned model off the twin; validating it there is future work.
- **The suffix check is symbolic.** g2 is a STRIPS walk over declared and learned effects, and g3 rehearses only up to the recovered step. Downstream steps are confirmed by live execution and the end-of-turn goal check, not rehearsed.
- **Code grounding, not hint-dependence, is the dominant limiter.** Models often name the right concept but write it against accessors the scene lacks, or with forbidden tokens (Table 10). The physical test rejects every such predicate, so no mis-grounded predicate was ever accepted, but each one costs a round.
- **Per-scenario flags.** Compound edits ($K > 1$), skill witnesses, skill sequences, tail repair and predicate sets are enabled per scenario where the failure class needs them (Table 7); they are not all on by default.
- **Imperfect learned models under replanning.** The learned model can be subtly wrong (over-claimed effects, mis-scoped preconditions) and still yield a successful spliced repair (Appendix C.4). Such errors surface only when a later planner call trusts the model, and a post-hoc model audit using the same interventions is future work.

- **Best-of selection.** Headline ablation numbers select the best configuration per scenario. Pooled numbers are 54.5% (SMOOTH) over all $1{,}327$ comparable episodes. The prompt and observation ablations use one seed on small subsamples, and their differences should be read as directions rather than significant effects.