arXiv:2610.40016v1 [cs.RO] 29 Oct 2026

# LOGIV: Graph-Verified Execution with Causal Local Repair for Long-Horizon Robot Tasks

**Enqiao Lu[1,2,7,*], Yuxiang Xie[1,6,*], Xingrui Yu[2,†], Yuyang Wang[3,6], Yipeng Gu[1], Zhenglin Wan[4], Pengfei Zhou[4], Wangbo Zhao[5], Yang You[4], Yuxiang Gao[6], Ivor Tsang[2,7]**

[1]CUHKSZ, [2]A*STAR, [3]Caltech, [4]NUS, [5]HKUST, [6]COCO Matrix, [7]NTU
[*]Equal contribution, [†]Corresponding author

Agentic robots must turn a high-level goal into actions while keeping the physical conditions needed for future steps valid. A completed action does not ensure that its effects will persist: a later disturbance can invalidate support that the remaining plan still assumes. We present LOGIV, a graph engineering framework for agentic robot execution that uses an explicit task graph to connect symbolic planning, observation-based verification, and embodied action. LOGIV compiles a validated plan into condition-support and resource-ordering edges, checks their physical validity during execution, and uses dependency tracing to determine what must be restored after a failure. Repair is constrained by retained unfinished actions, and the combined continuation is symbolically validated before execution resumes. The framework operates above existing Vision-Language-Action (VLA) and World Action Model (WAM) policies without adding trainable components. Across 20 dual-arm tasks from RoboTwin V2 and BiCoord with three execution backbones, LOGIV improves mean task success without injected disturbances by 10.67 and 15.33 percentage points, respectively. Under predefined perturbations, it achieves higher task completion and condition restoration than three adapted recovery baselines for every backbone on both benchmarks. On three real-robot tasks, successful trials increase from 12/30 to 17/30. These results support explicit, continuously checked task dependencies as a practical interface for reliable agentic robot execution.

Email: 124090411@link.cuhk.edu.cn, yu_xingrui@a-star.edu.sg
GitHub: https://github.com/Enqiao-Lu/LOGIV
Project Page: https://enqiao-lu.github.io/logiv-project/

## I. Introduction

Agentic robots combine goal-directed planning, perception, and physical action in a feedback loop (Qin et al., 2026). Vision-Language-Action (VLA) models (Octo Model Team et al., 2024; Kim et al., 2024; Physical Intelligence et al., 2025) and World Action Models (WAMs) (Yuan et al., 2026; Cai et al., 2026) provide increasingly capable execution policies, while an agent-level controller must decide how their actions contribute to a longer task. A distinctive difficulty in physical execution is that an intermediate result can cease to hold after the action that produced it has succeeded. A support placed during bridge assembly may be displaced before the beam is installed (Fig. 1). The placement remains a completed action, yet the support condition needed by the remaining plan is no longer valid. Reliable long-horizon behavior therefore requires tracking both action completion and the continuing validity of its consequences.

Existing work provides important components for this execution loop. LLM- and VLM-based systems select skills, interpret feedback, and organize multistep behavior (Ouyang et al., 2025; Huang et al., 2024; Feng et al., 2025; Chen et al., 2025c; Ji et al., 2026; Peng et al., 2026a; Zhang et al., 2026b; Huang et al., 2026). Symbolic and program-based methods make action constraints explicit (Yang et al., 2025; Kwon et al., 2025; Chen et al., 2025b; Chu et al., 2025), building on established work in plan monitoring and repair (Pollack and McCarthy, 1999; Van Der Krogt and De Weerdt, 2005). Graph-based agent frameworks expose subtask dependencies for reuse and repair (Zhang et al., 2026a); in robotics, AgentChord augments task graphs with anticipatory recovery branches before execution (Xu et al., 2026). These advances raise a question: when observations invalidate an

earlier condition, which support should the robot restore while preserving a usable unfinished plan?

We approach this question through *graph engineering*: constructing, checking, and revising an explicit dependency graph as the interface between planning and robot execution. We present LOGIV (Long-Horizon Online Graph-based Inference and Verification for Plan Repair). Its task graph records which conditions each action supplies to later actions, together with resource-ordering constraints. The key design is to separate a node's completion status from the current validity of the support it provides. When that support fails, unfinished consumers can remain in the plan while repair re-establishes their prerequisites. The retained continuation then constrains the repair goal. This gives graph structure an operational role in deciding what to check, what to preserve, and what to change.

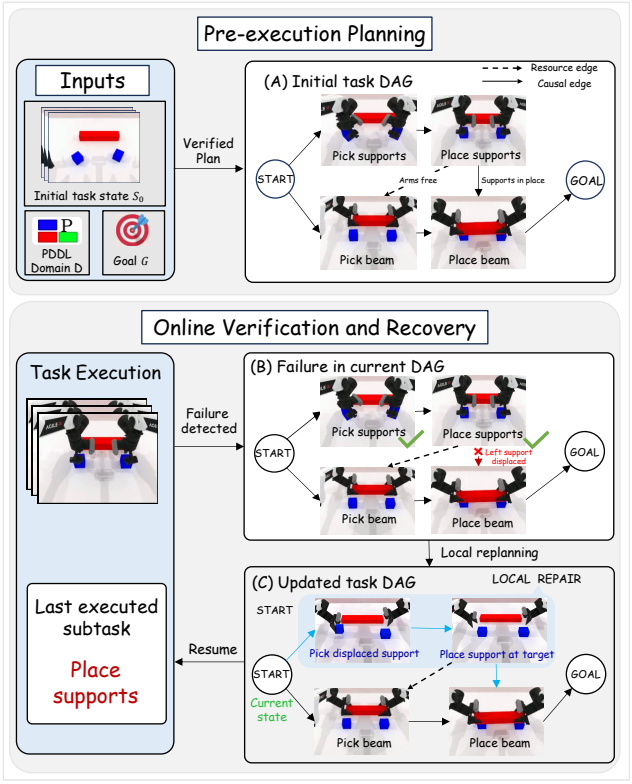



Fig. 1. LOGIV execution on the BiCoord `build_bridge` task. (A) A validated reference plan is compiled into the initial task DAG before execution. (B) A previously placed support is displaced, invalidating a condition required by the remaining plan. (C) LOGIV repairs only the affected support operations, retains the unaffected beam operations, and resumes execution on the updated task DAG.

LOGIV implements this interface as a closed-loop execution and repair layer above an existing VLA or WAM policy. Before execution, it validates a high-level plan and compiles its dependencies into a task graph. During execution, visual observations and robot feedback check the selected action and the support required by pending actions. After a confirmed failure, dependency tracing identifies affected actions; repair synthesizes a continuation constrained by retained operators and expands the replacement scope when validation fails. The combined plan is checked under the symbolic model before the graph is rebuilt and execution resumes. In the bridge example, the robot restores the displaced support while retaining the remaining beam operations. This design adds no trainable components and focuses on online repair from the observed state, with the validity of the repaired continuation checked explicitly. We evaluate LOGIV on 20 simulated dual-arm tasks from RoboTwin V2 and BiCoord using $\pi_{0.5}$, AHA-WAM, and Light-WAM. Without injected perturbations, mean success across the three backbones increases from 71.67% to 82.33% on RoboTwin V2 and from 42.33% to 57.67% on BiCoord. Under predefined perturbations, LOGIV achieves higher task completion and condition restoration than three adapted recovery baselines for every backbone on both benchmarks. On three real-robot tasks, successful trials increase from 12/30 to 17/30. Ablations support the role of graph structure: removing dependency checks reduces perturbed-task success from 60% to 48%, and replacing local repair with replanning all remaining actions increases repeated valid actions from 0.2 to 1.6. These results support graph engineering for the evaluated tasks. Our contributions are:

- A graph engineering formulation of agentic robot execution that separates completed actions from the continuing validity of their condition support during physical task execution.
- A closed-loop implementation combining observation-based graph checks, dependency-guided repair constrained by retained actions, and symbolic validation of the full continuation.
- Evaluation across two simulation benchmarks, three execution backbones, and real robots, with ablations examining how graph verification and repair design affect task success and redundant execution.

## II. Related Work

### A. Long-Horizon Manipulation with VLA and WAM

Recent approaches improve long-horizon manipulation through high-level execution organization or changes to the policy backbone. In the first group, a high-level module selects and sequences subtasks while a low-level policy executes them. HiMe uses hierarchical memory to preserve task context, Cortex aligns planning and execution interfaces, and Harness VLA and RoboHarness use execution memory to organize robot policies (Ji et al., 2026; Peng et al., 2026a; Zhang et al., 2026b; Huang et al., 2026). These methods motivate the use of an explicit execution layer around an existing policy. Other approaches improve the representation or use of temporal information within the model. RoboHorizon combines multiview world modeling with mechanisms for long-horizon task organization (Chen et al., 2025c). Fast-WAM studies whether future generation is needed at test time (Yuan et al., 2026), while AHA-WAM uses asynchronous world and action branches with observation-guided context routing (Cai et al., 2026). LOGIV addresses a complementary system-level problem: how to update a high-level plan when an execution error or external disturbance invalidates a condition needed later in the task.

### B. Plan Verification and Recovery

Structured representations can support both plan checking and online recovery. VeriGraph uses scene graphs to verify and refine candidate action sequences (Ekpo et al., 2024). Other graph-based systems maintain scene information or task structure for planning and execution (Shirasaka et al., 2025; Yang et al., 2026; Zhang et al., 2026a). LERa uses visual feedback to identify failures and revise a plan (Pchelintsev et al., 2025), Reflective Planning reassesses actions using predicted observations (Feng et al., 2025), and behavior-tree approaches provide reactive execution and recovery mechanisms (Ahmad et al., 2025; Colledanchise and Ögren, 2016; Iovino et al., 2022). Recent language-based systems use visual or multimodal feedback to detect execution failures, explain plan–execution mismatches, or trigger corrective replanning (Guo et al., 2024; Zhou et al., 2025; Liu et al., 2023b; Duan et al., 2024; Skreta et al., 2024; Huang et al., 2022). Symbolic planning provides explicit preconditions and effects that can be checked before execution (Yang et al., 2025; Kwon et al., 2025; Chen et al., 2025b; Chu et al., 2025; Liu et al., 2023a). Causal-link monitoring and plan repair provide a basis for reasoning about support relations from executed actions to pending actions. LOGIV builds on these ideas rather than treating dependency tracking itself as a new planning principle. It specifically addresses support conditions established by completed subtasks that later become invalid, using visual state assessment to update explicit producer–consumer dependencies before selecting a local repair scope. Retained actions constrain the continuation, and the combined plan is validated before execution resumes. The experiments compare this framework with adapted verification and recovery baselines under a shared evaluation protocol.

## III. Method

LOGIV separates high-level task management from low-level control: it constructs, checks, and repairs plans, while the execution policy carries out individual subtasks. We use bridge assembly as a running example: the robot places supports, grasps a beam, and lays it across them.

### A. Problem Formulation

At round $t$, $O_t$ is the visual observation and $S_t$ is the symbolic state estimated from visual interpretation, robot status, and actuator feedback. Given $S_0$ and goal $g$, the Operator Planner generates a reference plan $P = (o_1, \ldots, o_n)$ of grounded PDDL operators with specified preconditions and effects (McDermott et al., 1998). VAL checks the plan (Howey et al., 2004), which is then compiled into a directed acyclic graph $G = (V, E)$.

$$\begin{aligned} P &= \mathrm{PreparePlan}(S_0, g) = (o_1, \ldots, o_n), \\ G &= (V, E) = \mathrm{BuildGraph}(S_0, P, g). \end{aligned} \tag{1}$$

The nodes in $V$ are subtasks, START, and GOAL. Causal edges record which action supplies another's required condition; resource edges encode ordering constraints from resource use or modeled interference. Here, causal means PDDL condition support, not a learned model. The reference plan orders the graph and breaks scheduling ties.

The selected operator is passed to the low-level policy as a natural-language instruction with the current visual observation. After execution, LOGIV updates the state and checks the outcome and dependencies. A confirmed execution failure produces an error record $e_t$ and triggers plan repair:

$$(P', G') = \text{TopologyGuidedRepair}(S_{t+1}, e_t, P, G). \tag{2}$$

The updated plan combines a repair fragment with retained unfinished operators. Validation from $S_{t+1}$ establishes consistency with the symbolic model and supplied state, without guaranteeing perception accuracy or physical success. Completed actions' valid effects remain in the current state; the completed prefix is not replayed.

### *B. Pre-Execution Planning*

**Plan generation and validation.** The Operator Planner receives a PDDL Domain and Problem (Fig. 2). The Domain specifies reusable operators, parameters, preconditions, and effects; the Problem specifies objects, initial state $S_0$, and goal $g$ (McDermott et al., 1998). LAMA-first (Richter and Westphal, 2010) in Fast Downward (Helmert, 2006) searches for a grounded sequence reaching $g$ from $S_0$. VAL independently checks action bindings, preconditions, modeled effects, and the final goal (Howey et al., 2004). On failure, it returns the location and reason. The Domain, objects, $S_0$, and $g$ remain unchanged; the failed sequence is excluded before searching for an alternative grounding or ordering. Graph construction begins only after validation passes.

**Task graph construction.** Each operator becomes a subtask node. For each precondition of $o_i$, the compiler searches preceding operators in reverse order for the nearest producer without an intervening invalidation in $P$. A causal edge links that producer to $o_i$ and is labeled with the supported condition. Conditions supplied by $S_0$ originate at `START`; goal conditions link to `GOAL` by the same rule. Placing a bridge support establishes the support-position condition needed by beam placement.

Resource edges retain ordering for exclusive resources, such as an arm or gripper, and modeled interference. Our PDDL uses typed object parameters with deterministic logical preconditions and effects. Operators interfere when one invalidates a condition required or established by the other. Both edge types follow $P$, making the graph acyclic and $P$ a topological order. Other topological orders need not be valid; execution order is checked online.

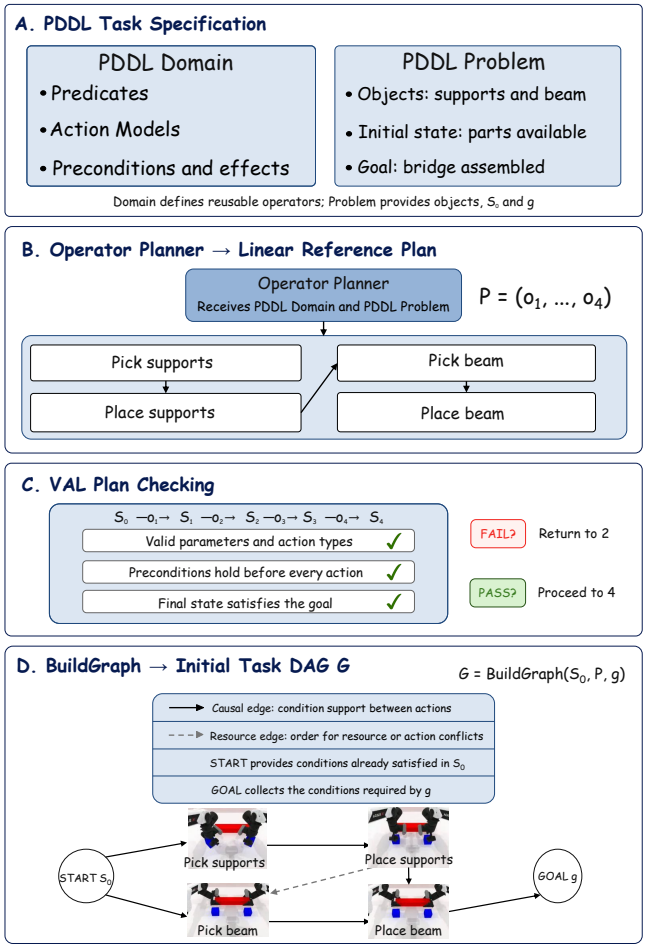



Fig. 2. LOGIV pre-execution planning and task-graph construction. The Operator Planner generates the reference plan $P$, where each $o_i$ in panel (B) is a grounded high-level PDDL operator. After validation by VAL, $P$ is compiled into the initial task graph $G$.

### *C. Online Verification and Recovery*

**Task-graph scheduling and online checking.** As shown in Fig. 3, the system schedules actions using causal dependencies and resource order in $G$. Among unfinished actions whose predecessors are complete, it selects the earliest action in $P$ whose execution conditions hold. If none is ready, it selects the earliest action with completed predecessors, refreshes the state, and checks whether repair is needed. Once all actions are complete, it checks $g$.

Before execution, the VLM extracts facts from $O_t$ about the current action and its dependencies. The State Gate combines them with robot state and actuator feedback to form $S_t$. If evidence is insufficient, the system acquires

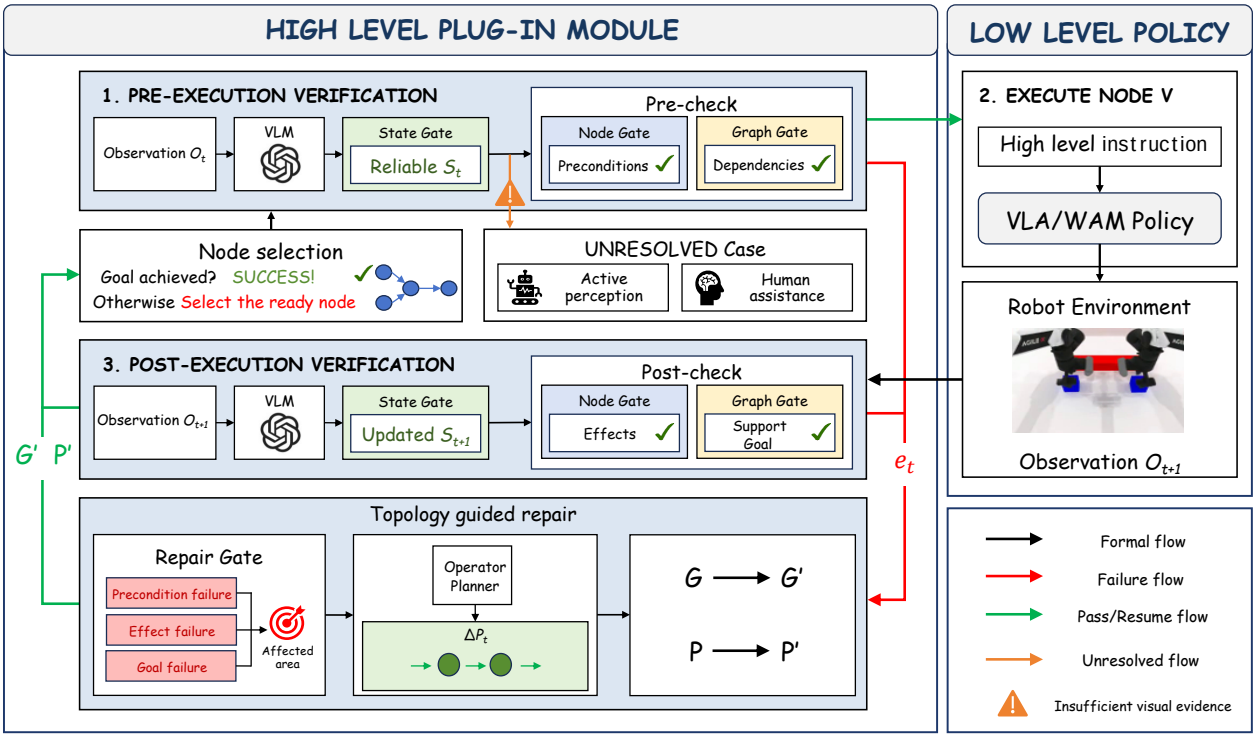



Fig. 3. LOGIV online verification and topology-guided causal local repair. The three gates establish reliable task facts, verify the selected subtask, and check graph dependencies. Upon a failure, the Repair Gate identifies affected nodes, validates a local repair plan, rebuilds the task graph, and resumes execution. Human assistance is reserved for future evaluation.

new images, adjusts the viewpoint, or repeats checks. Facts still unconfirmed within the observation budget yield `UNRESOLVED` and stop execution. The Node Gate checks preconditions; the Graph Gate checks resource predecessors and condition support. Confirmed unmet preconditions yield `FAIL` with `PRECONDITION_FAILURE`. Otherwise, the operator is converted into an instruction for the low-level policy.

After execution, the system updates $S_{t+1}$ from $O_{t+1}$ and robot feedback. The Node Gate checks the expected effects and marks the node complete if they pass. The Graph Gate then checks whether support from `START` and completed actions still holds for remaining actions and `GOAL`. Missing effects or invalidated support yield `FAIL` with `EFFECT_FAILURE`. Thus, a support displaced while grasping the beam triggers repair even if the grasp succeeds; the completed support-placement node is not reopened. If both checks pass, the system returns `PASS`. At `GOAL`, a verified $g$ yields success; a confirmed unmet goal yields `FAIL` with `GOAL_FAILURE`. Algorithm 1 in Appendix A gives the execution loop, with these checks summarized as

$$\begin{aligned}(\texttt{status}, S_{t+1}, e_t) &= \text{VerifyAndExecute}(v, G, S_t),\\ \texttt{status} &\in \{\texttt{PASS}, \texttt{FAIL}, \texttt{UNRESOLVED}\}\end{aligned} \tag{3}$$

The checked node is $v$. Record $e_t$ gives the failure type, node, unmet condition, and failed edge; it is empty otherwise. Completion status distinguishes missing action effects from a completed action's invalidated support.

**Repair scope selection.** Algorithm 2 in Appendix A distinguishes affected actions from those that must be replaced. After a failure, we identify the unfinished action at which the failure was detected, or the unfinished actions that directly depend on invalidated support, together with their unfinished causal successors. Initially, only a just-executed action whose own effects failed is replaced. An action with a missing precondition is retained, and repair restores that condition before execution. If support from a completed action is later invalidated, the action remains complete and its unfinished consumers are retained. A final-goal failure starts with an empty replacement set. In the beam example, beam placement remains in the plan while repair restores its required support.

**Repair planning and validation.** Once the replacement scope is fixed, the remaining unfinished actions form the retained plan $P_{\text{keep}}$. The system works backward from the final goal $g$ through these actions to determine what

must hold before their execution resumes; this result is denoted by $g_{\text{rep}}$. For actions with conjunctive preconditions and goals and unconditional add/delete effects, conditions are represented as signed literals. The system processes the retained actions in reverse order, starting from $g$. If an action deletes a condition that is still required, the requirement is infeasible; otherwise, the system removes the conditions established by that action and adds its preconditions. Complementary literals also make the requirement infeasible and cause planning to fail immediately. If no actions are retained, the repair fragment must directly achieve $g$. For example, a retained beam-grasp action requires `gripper_empty` and establishes `holding(beam)` for placement. The repair therefore needs an empty gripper before the grasp, but need not establish `holding(beam)` in advance. Starting from $S_{t+1}$, the Operator Planner generates a repair fragment $\Delta P_t$ that achieves $g_{\text{rep}}$, giving $P' = \Delta P_t \oplus P_{\text{keep}}$. The system uses VAL to check whether the complete plan $P'$ reaches $g$. A failure inside the repair fragment triggers replanning within the same scope, while a repeated candidate terminates repair. A failure at a retained action expands the replacement scope to that action and its unfinished causal successors. A planning failure, timeout, or unlocated failure invokes the deterministic fallback in the Notes. After validation succeeds, LOGIV rebuilds the graph from $P'$: repair actions become new nodes, completed actions remain complete, and their still-valid effects are represented by the current state and the new `START` node. Execution resumes from the earliest ready node in reference order. Repair terminates when no further expansion is possible or the budget is exhausted, without guarantees of minimality or completeness.

## IV. Experiments

### A. Experiment Setup

**Tasks and perturbations.** We evaluate LOGIV on ten tasks from RoboTwin V2 (Chen et al., 2025a) and ten from BiCoord (Peng et al., 2026b). Both benchmarks support dual-arm manipulation with multistage actions and cross-arm dependencies. Appendix B lists the selected tasks in Table A1. Each task has ten predefined initial states. The clean setting evaluates each state once without an injected disturbance. The perturbed setting reuses the same initial states and introduces one recoverable disturbance at a predefined event. The disturbance type and trigger are fixed within each task and vary across tasks. For example, in `move_can_pot`, the pot is shifted laterally by 10–15 cm after the can is grasped and lifted.

We design three real-world tasks (Fig. 4). In `pick_tomato`, the robot moves a tomato from the table into a bowl, a short task with few conditions. In `build_blocks`, both arms build a small block structure, where earlier placements establish support conditions for later ones. In `insert_cable`, the robot plugs an RJ45 connector into a switch port, requiring millimeter-level alignment beyond pick-and-place. No disturbances are injected in these trials.

**Backbones and baselines.** We use $\pi_{0.5}$ (Physical Intelligence et al., 2025), AHA-WAM (Cai et al., 2026), and Light-WAM (Li et al., 2026) as execution policies. For RoboTwin V2 and BiCoord, each backbone undergoes task-specific fine-tuning on the ten selected tasks in that benchmark. The backbone training or adaptation uses expert demonstrations without the evaluation-time perturbations. LOGIV itself introduces no additional trainable components. For each backbone, we compare LOGIV with the backbone alone and three adapted recovery methods. VeriGraph-PDDL follows the scene-graph verification approach of VeriGraph (Ekpo et al., 2024) and replans the remaining task from the current symbolic state when a constraint is violated. LERa-PDDL uses visual feedback to analyze a failure, updates the planning conditions, and replans the remaining task (Pchelintsev et al., 2025). VLM-RP-BT organizes the initial plan as a behavior tree (Colledanchise and Ögren, 2016; Iovino et al., 2022) and inserts recovery actions when a required condition is not satisfied. These names refer to the adaptations evaluated here, rather than to unmodified implementations of the original systems.

**Implementation details.** The experimental PDDL specification is predefined and held fixed. The VLM configuration uses GPT-4o with a maximum of 512 output tokens per call. Each node receives one pre-execution and one post-execution check. Human assistance is disabled in all reported experiments; potential failures first trigger active perception by the robot. An unresolved assessment allows at most three additional observations. We use LAMA-first in Fast Downward with a 30 s limit per planning call and a 5 s limit per VAL call. Each failure allows at most three local repair rounds and four VLM calls. Exceeding either limit terminates the episode.

For these tasks, we compare the base $\pi_{0.5}$ policy with $\pi_{0.5}$ + LOGIV on two physical dual-arm platforms: an Astribot S1 whole-body dual-arm robot and a workcell built from two AgileX Piper 6-DoF arms with parallel-jaw grippers. Each platform provides a third-person RGB-D camera and a wrist camera per arm. The baseline and

LOGIV configurations use the same $\pi_{0.5}$ checkpoint and the same demonstrations. LOGIV adds no trainable components. Each configuration is evaluated over 10 trials per task under a shared protocol with matched initial layouts. A trial counts as successful only when the full instruction is completed.

**Evaluation metrics.** We report full-task success without injected perturbations (C), full-task success with injected perturbations (P), and condition restoration (R). An event is eligible if execution reaches the injection point, the disturbance invalidates the designated condition, and recovery remains possible with the available action set. Restoration succeeds if the condition is restored within budget and passes the next relevant check. For each method, $R$ is the fraction of eligible events with successful restoration, counting failures in the denominator. Event sets may differ across methods, so $R$ is read alongside $P$. The ablation study additionally reports the fraction of unexecuted nodes deleted, replaced, or replanned during repair, and the number of still-valid completed actions executed again. These measures describe plan changes and redundant execution, respectively.

### *B. Detailed Results*

We report results from simulation benchmarks and real-world robot experiments.

TABLE I

Task completion and condition restoration across backbones and online correction methods.

| Backbone | Method | RoboTwin V2 | | | BiCoord | | |
|---|---|---|---|---|---|---|---|
| | | C ↑ | P ↑ | R ↑ | C ↑ | P ↑ | R ↑ |
| $\pi_{0.5}$ | Backbone | 56 | 32 | 18 | 39 | 31 | 21 |
| | VeriGraph-PDDL | 68 | 56 | 66 | 46 | 39 | 56 |
| | LERa-PDDL | 67 | 57 | 64 | 47 | 43 | 63 |
| | VLM-RP-BT | 63 | 58 | 68 | 45 | 45 | 59 |
| | LOGIV | **72** | **60** | **72** | **50** | **48** | **68** |
| AHA-WAM | Backbone | 85 | 60 | 35 | 45 | 43 | 37 |
| | VeriGraph-PDDL | 90 | 78 | 81 | 56 | 46 | 77 |
| | LERa-PDDL | 88 | 79 | 82 | 52 | 48 | 73 |
| | VLM-RP-BT | 91 | 80 | 80 | 57 | 48 | 80 |
| | LOGIV | **93** | **83** | **85** | **62** | **53** | **84** |
| Light-WAM | Backbone | 74 | 58 | 28 | 43 | 39 | 31 |
| | VeriGraph-PDDL | 76 | 68 | 65 | 56 | 46 | 67 |
| | LERa-PDDL | 76 | 67 | 70 | 51 | 45 | 65 |
| | VLM-RP-BT | 78 | 70 | 62 | 58 | 43 | 68 |
| | LOGIV | **82** | **72** | **79** | **61** | **49** | **74** |

**C**: Task success rate without artificial perturbations; **P**: Task success rate after applying predefined perturbations; **R**: Recovery success rate for invalidated conditions among valid, recoverable perturbation events. All results are expressed as percentages (%).

**Simulation experiments.** Table I summarizes the results of LOGIV on both benchmarks. Without artificial perturbations, integrating LOGIV raises mean task success across the three backbones from 71.67% to 82.33% on RoboTwin V2 and from 42.33% to 57.67% on BiCoord. Robots can still encounter state deviations and action failures in these runs. The gains therefore show that LOGIV improves normal task execution as well as recovery from injected disturbances. Under predefined execution-time perturbations, LOGIV also improves task completion. On RoboTwin V2, task success with $\pi_{0.5}$, AHA-WAM, and Light-WAM reaches 60%, 83%, and 72%, respectively, exceeding the corresponding backbone results by 28, 23, and 14 percentage points. On BiCoord, the respective success rates are 48%, 53%, and 49%, with gains of 17, 10, and 10 percentage points. LOGIV achieves higher task success than all three correction baselines in both settings, indicating that its benefits extend across backbones and benchmarks. The perturbed results further show that the framework can help execution continue after an external disturbance. We also evaluate condition restoration separately from full-task completion. LOGIV achieves higher

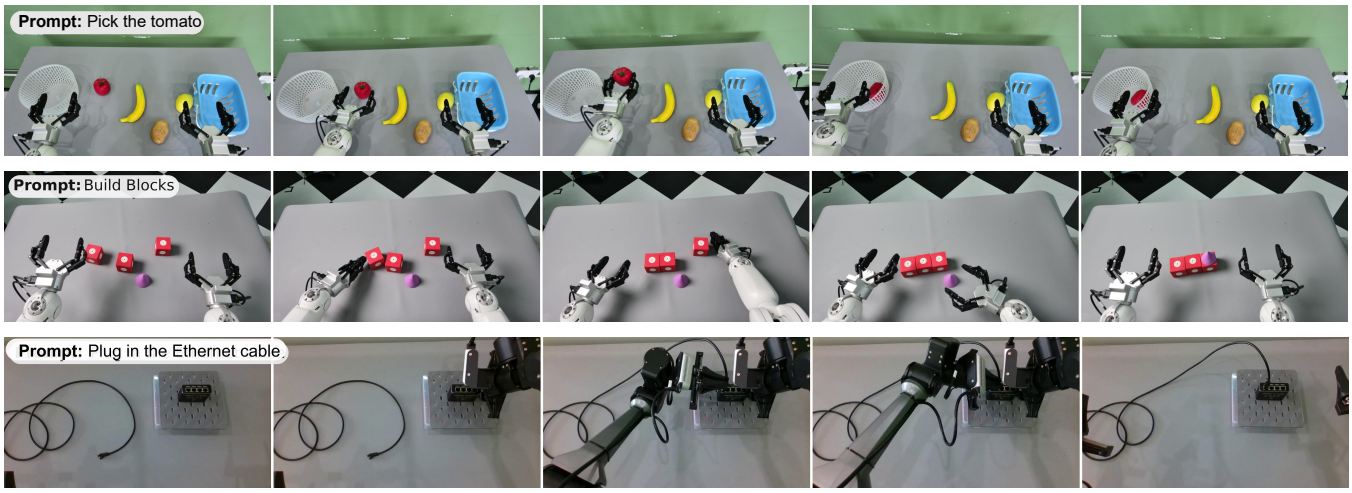



Fig. 4. Real-robot rollouts of the three tasks. Top: pick the tomato and place it in the bowl. Middle: build the block structure with both arms. Bottom: plug in the Ethernet cable.

recovery success than the three correction baselines for every combination of benchmark and backbone, showing consistent gains in restoring invalidated conditions within the evaluation budget. These results support the use of dependency checks and plan repair to help execution continue after a failure. To guide repair, LOGIV traces the conditions needed by subsequent actions, extending its assessment beyond the failed action itself. Retained actions constrain the updated plan, limiting changes to still-usable parts of the unfinished task.

TABLE II

Real-robot success over ten trials per task.

| Task | Platform | $\pi_{0.5}$ | $\pi_{0.5}$ + LOGIV |
|---|---|---|---|
| `pick_tomato` | Astribot S1 | 7 / 10 | **9 / 10** |
| `build_blocks` | Astribot S1 | 3 / 10 | **5 / 10** |
| `insert_cable` | AgileX Piper | 2 / 10 | **3 / 10** |
| Overall | — | 12 / 30 (40.0%) | **17 / 30 (56.7%)** |

**Real-world experiments.** Table II reports the real-robot results. LOGIV succeeds in 17 of the 30 trials compared with 12 for the base policy, raising the overall success rate from 40.0% to 56.7%. It improves performance on all three tasks. On `pick_tomato`, successes increase from 7 to 9, with recovery from empty grasps or early releases of the tomato. On `build_blocks`, successes increase from 3 to 5. This gain comes from restoring condition support: when a later reach displaces a previously placed block, the Graph Gate detects that a condition required by the remaining placements no longer holds. Repair restores that block while retaining placements that remain valid. On `insert_cable`, successes increase only from 2 to 3. Most residual failures occur when the plug is aligned but fails to seat, so a successful retry depends on the contact-rich precision of the low-level policy rather than on the plan. With ten trials per task, wide confidence intervals indicate a trend rather than a precise effect size.

### *C. Ablation Study*

We evaluate each ablation variant over 100 episodes per setting on the ten RoboTwin V2 tasks in Appendix B (Table A1), using $\pi_{0.5}$ with shared visual inputs, VLMs, and planning budgets.

**Execution verification.** Removing the State Gate, Node Gate, or Graph Gate lowers all three success measures. Removing the Graph Gate gives the lowest perturbed-task success, 48% compared with 60% for the full framework. This result is consistent with a benefit from checking dependencies beyond the selected action. Removing uncertainty handling lowers condition restoration from 72% to 59%.

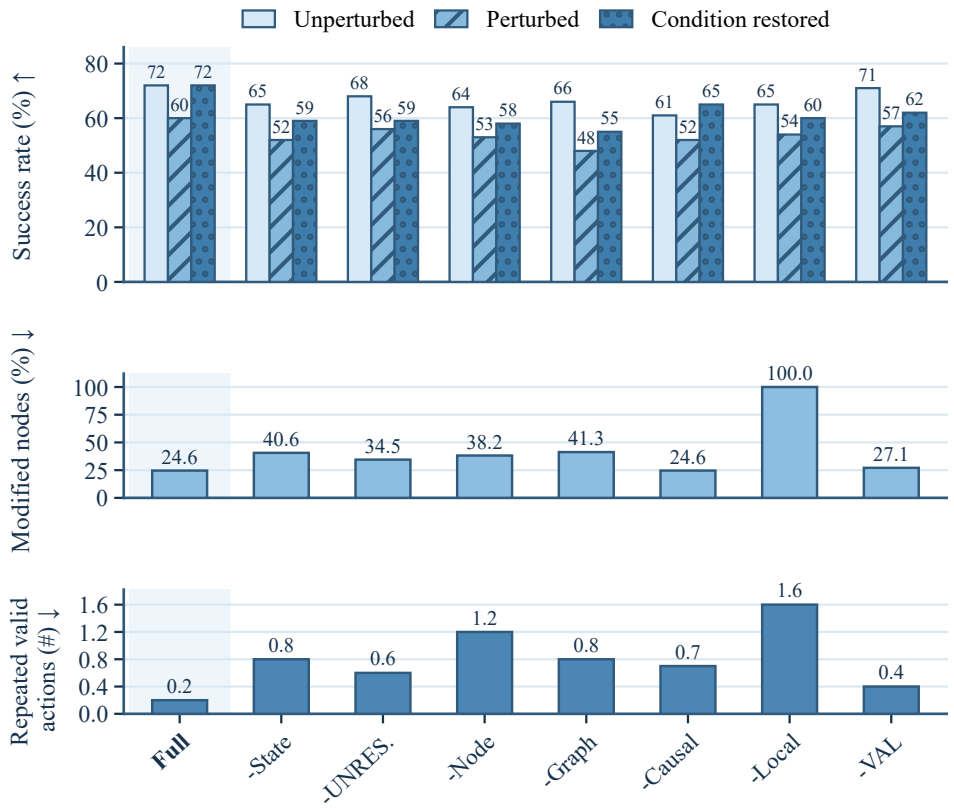



Fig. 5. Ablation results of LOGIV. The top panel shows task completion and condition restoration under perturbations. The middle and bottom panels show modified nodes and repeated valid actions during repair, respectively. Full: complete method; -State: no State Gate; -UNRES: no uncertainty handling; -Node: no Node Gate; -Graph: no Graph Gate; -Causal: no causal dependency tracing; -Local: replanning all remaining actions instead of local repair; -VAL: no logical validation of the repaired plan.

**Dependency tracing and local repair.** The variant without causal tracing selects repair nodes by their positions in the linear plan. Its mean modified-node fraction remains 24.6%, but clean-task success drops from 72% to 61% and perturbed-task success from 60% to 52%. Thus, the size of the selected region alone does not explain the result; the choice of nodes also matters. Replanning all remaining actions increases the modified-node fraction to 100% and repeated valid actions from 0.2 to 1.6, while perturbed-task success decreases to 54%. The modified-node fraction partly reflects this variant's definition and should be read alongside task success and repeated execution.

**Plan validation.** Without validation of the repaired plan, condition restoration decreases from 72% to 62%, while perturbed-task success decreases from 60% to 57%. VAL checks whether the repair fragment is compatible with retained operators and whether the complete continuation reaches the goal under the symbolic model. These results are consistent with a benefit from checking the combined plan rather than accepting a local fragment in isolation.

## V. Conclusion

We presented LOGIV, a framework that connects symbolic task dependencies with observation-based execution checks and local plan repair. After detecting an invalidated condition, LOGIV constructs a repaired continuation while retaining usable unfinished actions and validates the result before execution resumes. Experiments on RoboTwin V2 and BiCoord show higher observed task success across three execution backbones, while the number of successful real-robot trials increases from 12/30 to 17/30. Ablations are consistent with benefits from dependency tracing, local repair, and validation. The findings are limited to the evaluated tasks, perturbations, and trial counts. Future work will examine broader disturbances, incomplete state information, and the runtime and resource costs of online recovery. We will also evaluate human assistance in failure recovery.

## REFERENCES


F. Ahmad, H. Ismail, J. Styrud, M. Stenmark, and V. Krueger. A unified framework for real-time failure handling in robotics using vision-language models, reactive planner and behavior trees. In *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*, pages 887–894. IEEE, 2025.

J. Cai, L. Ling, S. Chu, Z. Liu, J. Kang, Z. Liang, W. Xu, Y. Mao, W. Zhang, X. Yang, et al. Aha-wam: Asynchronous horizon-adaptive world-action modeling with observation-guided context routing. *arXiv preprint arXiv:2606.09811*, 2026.

T. Chen, Z. Chen, B. Chen, Z. Cai, Y. Liu, Z. Li, Q. Liang, X. Lin, Y. Ge, Z. Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025a.

Y. Chen, Y. Hao, Y. Zhang, and C. Fan. Code-as-symbolic-planner: Foundation model-based robot planning via symbolic code generation. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 19248–19254. IEEE, 2025b.

Z. Chen, J. Huo, Y. Chen, and Y. Gao. Robohorizon: An llm-assisted multi-view world model for long-horizon robotic manipulation. *arXiv preprint arXiv:2501.06605*, 2025c.

K. Chu, X. Zhao, C. Weber, and S. Wermter. Llm+ map: Bimanual robot task planning using large language models and planning domain definition language. *arXiv preprint arXiv:2503.17309*, 2025.

M. Colledanchise and P. Ögren. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on robotics*, 33(2):372–389, 2016.

J. Duan, W. Pumacay, N. Kumar, Y. R. Wang, S. Tian, W. Yuan, R. Krishna, D. Fox, A. Mandlekar, and Y. Guo. Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation. *arXiv preprint arXiv:2410.00371*, 2024.

D. Ekpo, M. Levy, S. Suri, C. Huynh, A. Swaminathan, and A. Shrivastava. Verigraph: Scene graphs for execution verifiable robot planning. *arXiv preprint arXiv:2411.10446*, 2024.

Y. Feng, J. Han, Z. Yang, X. Yue, S. Levine, and J. Luo. Reflective planning: Vision-language models for multi-stage long-horizon robotic manipulation. *arXiv preprint arXiv:2502.16707*, 2025.

Y. Guo, Y.-J. Wang, L. Zha, and J. Chen. Doremi: Grounding language model by detecting and recovering from plan-execution misalignment. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12124–12131. IEEE, 2024.

M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

R. Howey, D. Long, and M. Fox. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE international conference on tools with artificial intelligence*, pages 294–301. IEEE, 2004.

J. Huang, Y. Hu, Z. Li, R. Qi, Y. Xiao, Z. Zhang, M. Coates, T. Cao, and Y. Zhang. Roboharness: Memory-driven orchestration of heterogeneous robot policies for long-horizon planning. *arXiv preprint arXiv:2607.18060*, 2026.

W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*, 2024.

M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith. A survey of behavior trees in robotics and ai. *Robotics and Autonomous Systems*, 154:104096, 2022.

L. Ji, S. Wang, P. Qian, X. Yu, Y. Tian, Z. Fei, J. Gong, and X. Qiu. Hime: Hierarchical embodied memory for long-horizon vision-language-action control. *arXiv preprint arXiv:2607.03449*, 2026.

M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

M. Kwon, Y. Kim, and Y. J. Kim. Fast and accurate task planning using neuro-symbolic language models and multi-level goal decomposition. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16195–16201. IEEE, 2025.

Z. Li, D. Cheng, Y. Wang, S. Wang, X. Xu, L. Weng, J. Wang, and J. Wang. Light-wam: Efficient world action models with state-fusion action decoding. *arXiv preprint arXiv:2606.08242*, 2026.

B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023a.

Z. Liu, A. Bahety, and S. Song. Reflect: Summarizing robot experiences for failure explanation and correction. *arXiv preprint arXiv:2306.15724*, 2023b.

D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL: The planning domain definition language, version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

Octo Model Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Y. Ouyang, J. Li, Y. Li, Z. Li, C. Yu, K. Sreenath, and Y. Wu. Long-horizon locomotion and manipulation on a quadrupedal robot with large language models. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11157–11164. IEEE, 2025.

S. Pchelintsev, M. Patratskiy, A. Onishchenko, A. Korchemnyi, A. Medvedev, U. Vinogradova, I. Galuzinsky, A. Postnikov, A. K. Kovalev, and A. I. Panov. Lera: Replanning with visual feedback in instruction following. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 19218–19225. IEEE, 2025.

J. Peng, X. Yu, D. Feng, Y. Yang, W. Cai, J. Xiong, G. Yang, J. Zheng, J. Cao, X. Wei, et al. Cortex: A bidirectionally aligned embodied agent framework for long-horizon manipulation. *arXiv preprint arXiv:2607.05377*, 2026a.

X. Peng, C. Gao, L. Jin, A. Li, and S. Liu. Bicoord: A bimanual manipulation benchmark towards long-horizon spatial-temporal coordination. *arXiv preprint arXiv:2604.05831*, 2026b.

Physical Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

M. E. Pollack and C. McCarthy. Towards focused plan monitoring: A technique and an application to mobile robots. In *Proceedings 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA'99 (Cat. No. 99EX375)*, pages 144–149. IEEE, 1999.

C. Qin, D. Wu, K. Feng, J. Wang, X. Yu, P. Sun, Y. Fan, X. He, B. Chen, W. Lian, R. Ji, W. Ren, X. Hu, D. Lin, Y.-G. Jiang, and X. Yue. Towards agentic robots: Model, data, environment, and harness. *arXiv preprint arXiv:2610.20442*, 2026.

S. Richter and M. Westphal. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

M. Shirasaka, Y. Ikeda, T. Matsushima, Y. Matsuo, and Y. Iwasawa. Spark: Graph-based online semantic integration system for robot task planning. *arXiv preprint arXiv:2506.20394*, 2025.

M. Skreta, Z. Zhou, J. L. Yuan, K. Darvish, A. Aspuru-Guzik, and A. Garg. Replan: Robotic replanning with perception and language models. *arXiv preprint arXiv:2401.04157*, 2024.

R. Van Der Krogt and M. De Weerdt. Plan repair as an extension of planning. In *ICAPS*, volume 5, pages 161–170, 2005.

S. Xu, R. Jin, H. Zhou, B. Yue, G. Qiao, Y. Tai, Y. Deng, K. Jia, and G. Liu. From reaction to anticipation: Proactive failure recovery through agentic task graph for robotic manipulation. *arXiv preprint arXiv:2605.11951*, 2026.

H. Yang, Z. Jiao, S. Wang, Y. Niu, S. Liu, and H. Liu. Integrated exploration and sequential manipulation on scene graph with llm-based situated replanning. *arXiv preprint arXiv:2602.04419*, 2026.

Z. Yang, C. Garrett, D. Fox, T. Lozano-Pérez, and L. P. Kaelbling. Guiding long-horizon task and motion planning with vision language models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16847–16853. IEEE, 2025.

T. Yuan, Z. Dong, Y. Liu, and H. Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026.

Y. Zhang, S. Chen, Z. Huang, H. Cui, K. Ji, and Z. Wang. Atomic task graph: A unified framework for agentic planning and execution. *arXiv preprint arXiv:2607.01942*, 2026a.

Y. Zhang, H. Zhang, F. Gao, X. Li, Z. Liu, C. Zhu, J. Qiu, Y. Yan, J. Liu, W. Tang, et al. Harness vla: Steering frozen vlas into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026b.

E. Zhou, Q. Su, C. Chi, Z. Zhang, Z. Wang, T. Huang, L. Sheng, and H. Wang. Code-as-monitor: Constraint-aware visual programming for reactive and proactive robotic failure detection. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6919–6929. IEEE, 2025.

## APPENDIX A
## EXECUTION AND REPAIR ALGORITHMS

This appendix provides the pseudocode for the graph execution interface described in Section III. The two procedures separate execution monitoring from plan repair: the first determines when the current graph can advance, and the second constructs a validated continuation after a confirmed failure.

### A. Task-Graph Execution

Algorithm 1 connects initial planning, graph-based scheduling, observation-based verification, and repair. It advances only when the selected action and its required support pass the checks; unresolved state estimates terminate the episode. Goal satisfaction is checked explicitly after all actions are complete.

**Algorithm 1 LOGIV Task-Graph Execution and Repair**

**Require** initial task state $S_0$, task goal $g$
**Output** SUCCESS or FAILURE

$P \leftarrow \text{PreparePlan}(S_0, g)$
**if** $P =$ FAIL **then return** FAILURE
$G \leftarrow \text{BuildGraph}(S_0, P, g)$; $t \leftarrow 0$
**while** execution budget remains **do**
  **if** all actions are complete **then** $v \leftarrow$ `GOAL`
  **else** $v \leftarrow \text{SelectNode}(G, P, S_t)$
  **if** $v = \varnothing$ **then return** FAILURE
  $(\texttt{status}, S_{t+1}, e_t) \leftarrow \text{VerifyAndExecute}(v, G, S_t)$
  **if** `status` $=$ `UNRESOLVED` **then return** FAILURE
  **if** $v =$ `GOAL` and `status` $=$ `PASS` **then return** SUCCESS
  **if** `status` $=$ `FAIL` **then**
    $(P', G') \leftarrow \text{TopologyGuidedRepair}(S_{t+1}, e_t, P, G)$
    **if** $P' =$ FAIL **then return** FAILURE
    $(P, G) \leftarrow (P', G')$
  **end if**
  $t \leftarrow t + 1$
**end while**
**return** FAILURE  // budget exhausted

*Notes.* SelectNode uses Section III-C's scheduling rule and returns $\varnothing$ if no unfinished action has completed predecessors; `START` counts as complete. VerifyAndExecute updates the state and $G$, marking the action complete once its effects pass, then checking remaining support.

### B. Topology-Guided Repair

Algorithm 2 distinguishes actions affected by a failure from actions that must be replaced. The retained unfinished actions determine the repair requirements through a backward pass. A candidate is accepted only after validation of the repair fragment together with the retained continuation; failures can expand the replacement scope within the stated budget. The procedure does not guarantee a minimal repair or completeness.

**Algorithm 2** **TopologyGuidedRepair**

**Require** symbolic state $S_{t+1}$, error $e_t$, plan $P$, graph $G$
**Output** repaired plan and graph $(P', G')$, or FAIL

$g \leftarrow \mathrm{Goal}(G)$; $(\mathcal{D}, \mathcal{A}) \leftarrow \mathrm{Locate}(e_t, G)$; $H \leftarrow \varnothing$
**while** repair budget remains **do**
  $P_{\mathrm{keep}} \leftarrow P_{\mathrm{unfinished}} \setminus \mathcal{A}$ // retained plan
  $g_{\mathrm{rep}} \leftarrow g$ // repair goal
  **for each** $a \in P_{\mathrm{keep}}^{\mathrm{rev}}$ **do**
    $g_{\mathrm{rep}} \leftarrow \mathrm{Backstep}(g_{\mathrm{rep}}, a)$
    **if** $g_{\mathrm{rep}} =$ FAIL **then return** FAIL
  **end for**
  $\Delta P_t \leftarrow \mathrm{OperatorPlanner}(S_{t+1}, g_{\mathrm{rep}})$; $u \leftarrow \varnothing$
  **if** $\Delta P_t \notin \{\mathrm{FAIL}, \mathrm{TIMEOUT}\}$ **then**
    **if** $\Delta P_t \in H$ **then return** FAIL
    $H \leftarrow H \cup \{\Delta P_t\}$; $P' \leftarrow \Delta P_t \oplus P_{\mathrm{keep}}$
    $(\mathtt{valid}, f) \leftarrow \mathrm{VAL}(S_{t+1}, P', g)$
    **if** `valid` **then return** $(P', \mathrm{BuildGraph}(S_{t+1}, P', g))$
    **if** $f \in \Delta P_t$ **then continue** // retry same scope
    $u \leftarrow \mathrm{EarliestFail}(f, P_{\mathrm{keep}})$
  **end if**; **if** $u = \varnothing$ **then** $u \leftarrow \mathrm{Fallback}(P_{\mathrm{keep}}, \mathcal{D})$
  **if** $u = \varnothing$ **then return** FAIL
  $\mathcal{A} \leftarrow \mathcal{A} \cup \mathrm{Succ}_G^*(u)$; $H \leftarrow \varnothing$
**end while**; **return** FAIL

*Notes.* $\mathcal{D}$ and $\mathcal{A}$ denote affected and currently replaced actions. Backstep performs the backward update in Section III-C and returns FAIL on conflict. $P_{\mathrm{unfinished}}$ follows the order in $P$; $P_{\mathrm{keep}}^{\mathrm{rev}}$ is $P_{\mathrm{keep}}$ in reverse. EarliestFail selects the earliest failing retained action. Fallback selects the first retained action in $\mathcal{D}$, otherwise the first in $P_{\mathrm{keep}}$, or $\varnothing$. $\mathrm{Succ}_G^*$ includes $u$ and its unfinished causal successors. Locate follows the scope-selection rules in Section III-C; BuildGraph follows Section III-B; $H$ stores current-scope candidates.

## APPENDIX B
## SIMULATION TASK SUITE

Table A1 lists the ten selected tasks from each simulation benchmark. These are the same tasks used for the clean and perturbed settings in Section IV-A; the ablations use the RoboTwin V2 subset. Each task has ten predefined initial states, giving 100 episodes per benchmark and setting. Task names are retained as benchmark identifiers to make the evaluation scope explicit. The table lists the selected subset, not the complete benchmark collections.

TABLE A1
THE 20 DUAL-ARM TASKS FROM ROBOTWIN V2 AND BICOORD.

| RoboTwin V2 (Chen et al., 2025a) | |
|---|---|
| `beat_block_hammer` | `blocks_ranking_size` |
| `handover_block` | `move_can_pot` |
| `open_microwave` | `place_dual_shoes` |
| `stack_blocks_three` | `stack_bowls_three` |
| `stamp_seal` | `turn_switch` |
| **BiCoord (Peng et al., 2026b)** | |
| `cook` | `put_objects_cabinet` |
| `divide_block_tower` | `place_plate_and_cup` |
| `build_bridge` | `clean_table` |
| `collect_pens` | `match_blocks_with_signs` |
| `jigsaw` | `exchange_mics` |