ROLLBACKNAV: PROGRAM-BASED ROLLBACK FOR RELIABLE EMBODIED NAVIGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Real-world embodied navigation differs from simulation in that robots operate under physical constraints that prevent instantaneous location reset. In practice, execution errors are inevitable, making recovery costly. However, existing approaches treat failure recovery implicitly, relying on heuristic backtracking or assumed state resets, rather than modeling recovery as an explicit execution capability. We propose **RollbackNav**, a program-guided navigation paradigm that elevates rollback to a first-class, physically executable operation during task execution. **RollbackNav** represents vision-language-navigation (VLN) behaviors as executable programs with explicit intermediate states, enabling step-level error localization, a structured rollback strategy, and validation of correctness. Based on this representation, we define a navigation-consistent rollback strategy with trajectory-aligned inverse operations, and introduce a program-guided recovery mechanism that selects rollback points based on semantic consistency and verifies rollback correctness via perceptual feature consistency. Experiments on VLN benchmarks, high-fidelity simulation, and real-world robotic deployments show that **RollbackNav** consistently outperforms heuristic-based rollback strategies, achieving more reliable recovery and higher rollback success. The code and project are available at: `https://anonymous.4open.science/r/RollbackNav-77DF/`

1 INTRODUCTION

With the rapid advancement of large-scale models, embodied agents have shown increasing capabilities in perception, decision-making, and action execution Fung et al. (2025); Salimpour et al. (2025); Li et al. (2025); Guan et al. (2025); Yang et al. (2025), enabling progress on complex tasks such as VLN in open-ended environments Wang et al. (2025); Ding et al. (2025); Zhang et al. (2025); Yue et al. (2025); Song et al. (2025); Qiao et al. (2025); Wang et al. (2024). Despite these advances at the algorithmic and model level, the reliable deployment of embodied intelligence in real-world physical settings remains a fundamental challenge. In particular, failure recovery during task execution has received limited attention. Unlike software systems or simulators, physical robots cannot be reset to an initial state at negligible cost; instead, navigation errors accumulate over long trajectories and re-deployment often requires substantial time, energy, and human intervention. As a result, execution failures in real-world environments incur disproportionately high downstream costs, revealing a critical gap between current embodied navigation research and practical, scalable deployment.

However, existing works often treat failure recovery in an implicit and oversimplified manner, summarized as follows. **(1) Outcome-only detection without execution semantics:** although actions and observations of VLN are generated step by step, most methods lack explicit, evaluable execution semantics during execution and therefore can only detect failures at the terminal outcome Perincherry et al. (2025); Liu et al. (2024); Wang et al. (2023), providing no principled mechanism to localize where errors arise during execution. **(2) State jump-back recovery without rollback tracking:** when failures occurred, most methods either ignore recovery or assume direct state reversion Qin et al. (2025); He et al. (2024), overlooking physical constraints. While rollback may be approximated via execution logs, these logs remain passive records. Replaying logged motions assumes deterministic and reversible dynamics, which rarely hold in real-world navigation, and thus cannot guarantee restoration of a semantically or physically consistent intermediate state. **(3) Heuristic tracking without correctness verification:** even when rollback actions are considered, methods such as heuristic tracking for a fixed number of steps cannot determine whether the robot has been correctly

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.

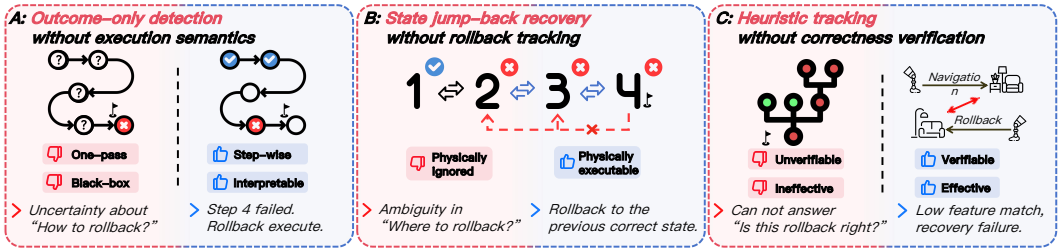


Figure 1: A comparison showing that existing methods suffer from missing step evaluations, state jumpbacks, and heuristic tracking, while **RollbackNav** eliminates these limitations.

restored to the intended position Ke et al. (2019); Shi et al. (2025). As a result, the correctness of recovery cannot be explicitly verified, often leading to repeated, unguided recovery attempts that ultimately result in excessive and ineffective actions. **Together**, these issues prevent robots from correctly and efficiently planning actions and executing strategic rollback in real-world settings (as shown in Figure 1). This prompts a key question: *Can we design a navigation paradigm with **(a)** perceptual error localization, **(b)** active rollback strategy and **(c)** precise recovery verification?*

Fortunately, programmatic/symbolic representations provide a tailored navigation paradigm, in which the code-pretrained model generates executable navigation code with explicit function composition and returned intermediate variables, making execution both verifiable, and interpretable. It is perfectly aligned with navigation requirement because: **(a)** the intermediate variables of the program, both perceptual and action-related, are inherently verifiable, allowing errors to be diagnosed based on faithful execution feedback rather than being deferred to the result; **(b)** the structured nature of program execution naturally provides explicit anchors for rollback, while the underlying syntactic and semantic constraints ensure that rollback code remains consistent with the original navigation logic; and **(c)** programs can invoke diverse APIs, such as vision, odometry, and inertial sensors, enabling multi-modal evaluation of state discrepancies before and after rollback to verify recovery correctness.

Motivated by programmatic representations, we introduce **RollbackNav**, a program-guided navigation paradigm that elevates rollback to a first-class, executable component of execution. We illustrate the workflow in Figure 2 and compare it with existing paradigms in the upper right. By grounding navigation in program execution, **RollbackNav** addresses the previous limitations by three key components: **(1) Interpretable navigation program generation**, which represents navigation tasks as a sequence of executable code generated by a code-pretrained model, and each code produces both robot actions and intermediate variables that encode perceptual feedback in execution for error detection. **(2) Navigation-consistent rollback strategy**, which strictly constrains rollback actions through code-level rules to remain consistent with the navigation action space, and improves rollback efficiency by merging homogeneous actions. **(3) Feature-based rollback recovery verification**, which verifies rollback correctness by leveraging keypoint-based image alignment techniques to determine whether the robot has returned to the intended location after rollback.

RollbackNav is a model-agnostic framework that leverages the program-generation capability of code-pretrained models to produce executable navigation programs with explicit execution and rollback constraints. Operating at the level of program structures, it can be readily integrated with diverse backbone models. We evaluate **RollbackNav** extensively on VLN tasks across standard benchmarks (R2R, RxR), a physically grounded simulator (VLN-CE-Isaac), and real-world robotic deployments. **RollbackNav** consistently improves navigation performance, achieves substantially higher recovery success rates, and further enhances dedicated VLN models when used as structured supervision. These results demonstrate the practical advantages of explicit, program-based rollback for reliable, real-world navigation. Overall, our contributions are summarized as follows:

- We propose **RollbackNav**, a program-guided navigation paradigm that enables perceptual error localization, active rollback strategy and precise recovery verification during real-world execution.
- We demonstrate how program-based representations support interpretable execution traces and structured rollback, providing a physically grounded alternative to recovery strategies in navigation.
- We conduct extensive evaluations on VLN benchmarks and real-world robotic deployments, showing that **RollbackNav** consistently outperforms strong baselines with minimal additional deployment overhead.

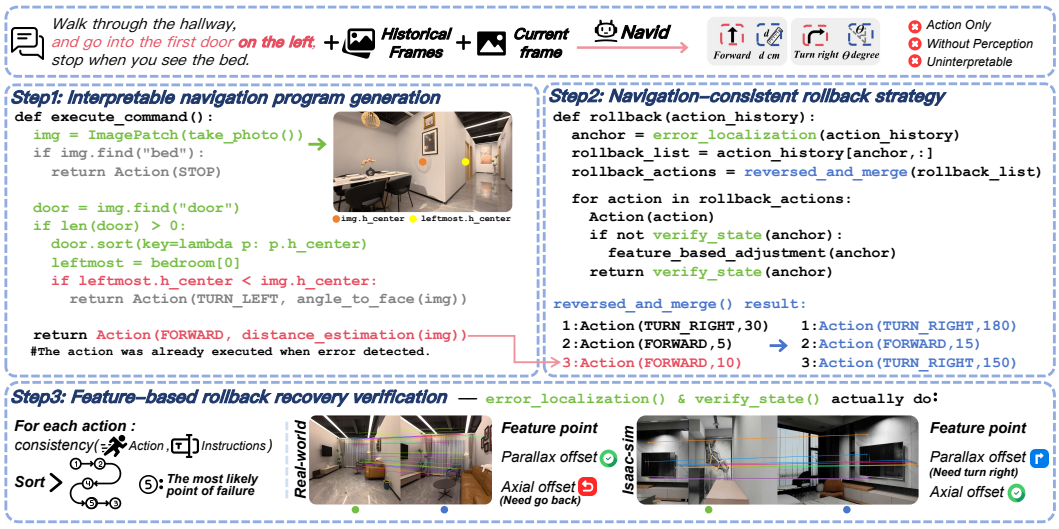


Figure 2: Overview of **RollbackNav**: Interpretable program generation provides step-level execution traces for error localization; a navigation-consistent rollback action space enables trajectory-aligned inverse operations without state jump-backs; and program-guided recovery evaluation verifies rollback correctness via perceptual feature consistency and resumes execution at intermediate states.

2 Related Work

2.1 Real-World Embodied Task Execution

Real-world operation on physical robots represents a key frontier for embodied intelligence Duan et al. (2022); Firoozi et al. (2025), where perception noise, actuation uncertainty, and dynamic scenes diverge from simulation and challenge conventional assumptions. As research increasingly shifts toward these settings, real-robot evaluations have emerged to assess navigation, manipulation, and language-conditioned behaviors in uncontrolled environments. However, most evaluations still focus on task-level episodic success Ahn et al. (2022); Driess et al. (2023), resolving failures through manual resets or heuristic replanning that implicitly assume low-cost intervention. This overlooks how agents could return to meaningful intermediate states and autonomously resume execution, leaving rollback underexplored as a first-class capability for real-world embodied task execution.

In contrast to these approaches, **RollbackNav** explicitly treats rollback as a physically executable and step-level process, enabling principled recovery and continuation without relying on manual intervention or oracle resets, which is essential for scalable and cost-effective real-world navigation.

2.2 Program-based Control for Embodied Agents

Recent work has explored representing agent behavior as executable programs rather than direct action sequences Huang et al. (2023); Zheng et al. (2023), offering improved interpretability and compositionality. In vision-language reasoning, approaches such as ViperGPT Surís et al. (2023) and Visual Programming Gupta & Kembhavi (2023) generate and execute code to compose modular operations, demonstrating the effectiveness of program-based representations for complex reasoning. In embodied and robotic settings, methods including Code as Policies Liang et al. (2022) and ProgPrompt Singh et al. (2023) further advocate programmatic interfaces that translate high-level language instructions into executable behaviors on real robots.

While these approaches demonstrate the benefits of program-based control for grounding and action composition, they mainly address forward execution and do not model rollback or post-failure recovery as physically executable processes. In contrast, **RollbackNav** uses program structure to make rollback a first-class capability, enabling step-level recovery and continuation essential for robust real-world navigation.

3 METHOD

We propose **RollbackNav**, a program-guided navigation paradigm for process-aware execution with physically executable rollback. Figure 2 overviews the framework. Given a navigation task, **RollbackNav** first generates an executable navigation program, where each statement is explicitly grounded in the agent's observations and actions, yielding step-level intermediate variables that enable perceptual error localization during execution (Section 3.1). Building on this programmatic representation, we define a navigation-consistent rollback action space with trajectory-aligned inverse operations, and enforce code-level constraints so that each executed motion admits a physically executable rollback counterpart (Section 3.2). Finally, we introduce a program-guided recovery mechanism that selects rollback points based on accumulated execution errors and verifies rollback correctness via perceptual feature consistency, enabling reliable resumption from principled intermediate states (Section 3.3).

3.1 INTERPRETABLE NAVIGATION PROGRAM GENERATION

In **RollbackNav**, we represent navigation task execution as an explicit program generation and execution process, in which agent actions are produced in an interpretable form. Programmatic representations provide explicit state dependencies and intermediate variables, enabling step-level execution monitoring and forming the foundation for process-aware execution and physical rollback.

Formally, let $\mathcal{T}$ denote a navigation task, and let $o_t \in \mathcal{O}$ represent the agent observation at time step t, including sensory inputs such as visual perception, proprioception, and task-specific state information. Given the task description $\mathcal{T}$ and the current observation o_t, a code-generating policy π_θ produces an executable program

$$z = \pi_\theta(\mathcal{T}, o_t), \tag{1}$$

where $z = [s_1, s_2, \ldots, s_n]$ is a sequence of program statements. Each statement s_i corresponds to a semantically meaningful operation grounded in perception, decision-making, or physical actuation.

The generated program z is executed by a program interpreter ϕ, which maps each program statement to a concrete executable action and updates the system state accordingly:

$$(a_i, o_{i+1}) = \phi(s_i, o_i), \tag{2}$$

where $a_i \in \mathcal{A}$ denotes the executable action at step i, and o_{i+1} is the resulting observation after execution. Sequential execution of the program induces an execution trace

$$\tau = \{(s_i, a_i, o_i)\}_{i=1}^{n}, \tag{3}$$

which explicitly records the correspondence between program statements, physical actions, and observed states.

Unlike end-to-end policies that directly map observations to actions, this programmatic formulation exposes intermediate computation and control flow at each execution step. Each program statement serves as a verifiable and interpretable unit, allowing downstream components to monitor execution progress, localize errors at the step level, and support physically executable rollback. This interpretable execution trace τ provides the foundation for defining rollback rules and principled recovery strategies, which we introduce in the following section.

3.2 NAVIGATION-CONSISTENT ROLLBACK STRATEGY

Rollback Trigger. Unlike conventional VLN systems that primarily generate actions from visual and linguistic observations, **RollbackNav** incorporates physical execution feedback to recognize navigation failures. Although signals such as collision, inability to make progress, or being stuck may be available from the underlying agent or simulator, existing methods typically lack a unified mechanism to interpret them as execution failures, relying on task- or environment-specific heuristics. In contrast, **RollbackNav** treats these failures as first-class signals: when a deviation from feasible navigation is identified, rollback is triggered to recover from the current execution state. Importantly, these signals determine *when* to invoke rollback, while the semantic consistency of historical execution steps determines *where* to roll back, as described in Section 3.3.

Table 1: Results on R2R and RxR validation unseen splits under different model baselines.

	Observation				R2R Val-Unseen					RxR Val-Unseen				
	Map	Odom.	Depth	S.RGB	NE↓	OS↑	SR↑	SPL↑	**RSR**↑	NE↓	OS↑	SR↑	SPL↑	**RSR**↑
Sim2Sim	✓	✓	✓		6.07	52.0	43.0	36.0	-	-	-	-	-	-
VLN⟳BERT	✓	✓	✓		5.74	53.0	44.0	39.0	-	8.98	-	27.0	22.6	-
LAW		✓	✓	✓	6.83	44.0	35.0	31.0	-	10.90	21.0	8.0	8.0	-
Seq2Seq			✓	✓	7.77	37.0	25.0	22.0	-	11.8	5.02	3.51	3.43	-
MapGPT (GPT-4)				✓	5.80	61.6	41.2	25.4	-	-	-	-	-	-
Navid				✓	5.47	49.1	37.4	35.9	-	8.41	34.5	23.8	21.2	-
Navila				✓	5.22	62.5	54.0	49.0	-	6.77	49.3	44.0	58.8	-
GPT-4 (NavGPT)				✓	6.46	42.0	34.0	29.0	-	8.39	34.8	24.5	22.6	-
- Prompt				✓	6.15	59.4	39.3	31.8	30.4	8.35	34.7	23.9	21.4	28.3
- Graph-based				✓	5.76	62.1	45.7	36.1	63.5	7.46	45.2	31.7	30.5	57.1
- **RollbackNav**				✓	5.28	63.8	57.4	42.6	88.9	7.23	49.6	38.2	36.1	82.4
Qwen2.5-VL-3B				✓	7.84	34.0	27.0	24.0	-	10.02	23.4	10.4	10.4	-
- Prompt				✓	7.51	39.5	31.9	27.3	21.5	9.34	28.0	10.1	9.8	19.6
- Graph-based				✓	6.63	43.4	38.2	34.7	54.7	8.71	33.9	21.6	20.7	52.6
- **RollbackNav**				✓	5.81	57.9	44.1	38.5	76.4	7.93	39.1	27.9	27.5	71.7

Rollback Space. Given a navigation program, rollback can be defined over its execution trace. We define the *rollback space* $\mathcal{R}$ as the set of executable action sequences composed exclusively of primitives from the original navigation action space. Given a current state o_t and a target rollback state o_k $(k < t)$, rollback seeks an executable sequence

$$\hat{a}_{k:t} = [\rho(a_t), \ldots, \rho(a_k)], \tag{4}$$

where $\rho(\cdot)$ denotes the rollback operator that compensates for an executed action. By restricting rollback to the original action space, **RollbackNav** avoids unrealistic state-jumping assumptions and keeps recovery physically grounded and consistent with the navigation process.

Action-level Rollback. For an executed action a_i, its rollback operation $\rho(a_i)$ compensates for the displacement or rotation induced by a_i. The concrete form of $\rho(a_i)$ depends on the underlying action space. For example, a forward translation of d meters may be reverted by a backward translation of d meters, or by a 180° rotation followed by a forward translation of d meters and a compensating rotation. To maintain trajectory consistency, consecutive rollback operations are merged only when they correspond to the same motion primitive. For example, two consecutive 15° rotations can be merged into a single 30° rotation, while two consecutive forward motions of 2 meters can be merged into a single 4-meter motion. Formally, for two consecutive rollback operations $\rho(a_i)$ and $\rho(a_{i-1})$, merging is permitted only if

$$\text{type}(\rho(a_i)) = \text{type}(\rho(a_{i-1})). \tag{5}$$

This constraint preserves trajectory alignment and reduces unnecessary deviations during recovery.

Overall, **RollbackNav** formulates rollback as an executable reversal of the program trace rather than a direct jump to a previously visited state. This formulation provides the basis for subsequent rollback-point selection and recovery verification in Section 3.3.

3.3 Feature-based Rollback Recovery Verification

Programmatic execution in **RollbackNav** constrains rollback to previously executed trajectories and well-defined intermediate program states. Under this setting, principled recovery requires answering two questions: *(i)* which program step execution should be rolled back to, and *(ii)* whether rollback correctly restores an intended intermediate state.

Rollback Point Selection via Semantic Inconsistency Ranking. **RollbackNav** localizes rollback points by analyzing semantic consistency along the execution trace. During forward execution, each program step i produces intermediate variables that encode perceptual outcomes and action semantics. When rollback is triggered, we compute a semantic consistency score $\mathcal{C}(i)$ for each historical step using the *speaker score* introduced in Speaker-Follower Fried et al. (2018), which measures the log-likelihood of generating the navigation instruction $\mathcal{I}$ conditioned on the executed trajectory $\mathcal{R}_i$:

$$\mathcal{C}(i) = \log P(\mathcal{I} \mid \mathcal{R}_i). \tag{6}$$



Figure 3: Two examples demonstrated in the Habitat and Isaac-Sim simulator, where the green trajectory represents the correct execution path, the red trajectory indicates an incorrect path, and the blue trajectory shows the corrected path after rollback.

A lower speaker score indicates that the executed trajectory is less consistent with the intended instruction, suggesting a larger semantic deviation from the desired navigation behavior. We therefore rank historical execution steps according to $\mathcal{C}(i)$ and identify the step with the lowest score as the most likely failure point. This allows **`RollbackNav`** to adaptively determine the rollback point based on execution semantics rather than relying on a fixed rollback length or heuristic step selection.

Rollback Correctness via Perceptual Consistency. After executing rollback actions to a candidate program step k, the agent re-executes the corresponding perception statements to obtain a new observation $\hat{o}_k$, which is compared against the original observation o_k recorded during forward execution. This ensures that verification is conditioned on identical program semantics. We evaluate perceptual consistency using FLANN-based keypoint matching implemented in OpenCV:

$$\mathcal{S}(\hat{o}_k, o_k) = \mathrm{Sim}\big(\mathrm{FLANN}(\mathcal{F}(\hat{o}_k), \mathcal{F}(o_k))\big), \tag{7}$$

where $\mathcal{F}$ extracts local keypoints and their corresponding descriptors; Sim measures the ratio of matched feature points. If $\mathcal{S}(\hat{o}_k, o_k) > \delta$, rollback is considered successful; otherwise, corrective adjustment is performed based on the spatial distribution of the matched feature points. For example, if the matched feature points are shifted to the left relative to the reference observation, the agent turns left accordingly, while a more concentrated distribution prompts the agent to move forward.

Together, these mechanisms determine *where* to roll back and *whether* rollback has been correctly executed, enabling reliable resumption from valid intermediate program states.

4 Experiments

To evaluate **`RollbackNav`**, we conduct experiments on standard VLN benchmarks (R2R and RxR), the physically grounded VLN-CE-Isaac simulator, and real-world robots (Sections 4.2–4.4). We further evaluate recovery reliability via Recovery Success Rate, the supervisory value of RollbackNav-generated trajectories, and key design choices through ablations (Sections 4.5–4.7).

4.1 Experimental Setup

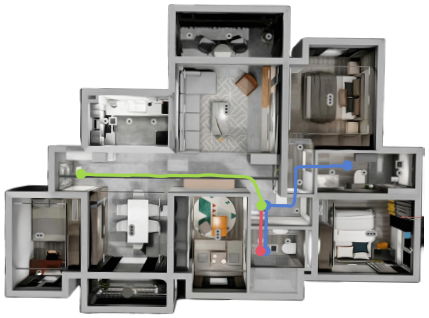
Model Setup. In the program generation stage, we adopt program-generation paradigms inspired by ViperGPT Surís et al. (2023) and Visual Programming Gupta & Kembhavi (2023), while modifying the API definitions to align with navigation execution and physically executable rollback. Detailed API specifications and prompting strategies are provided in the Appendix. For the rollback space, we consider a discrete action set consisting of four navigation primitives: *forward*, *rotate left*, *rotate right*, and *stop*.

For perceptual verification during rollback, we set the feature-matching threshold for perceptual consistency to 0.9. We evaluate **`RollbackNav`** using an open-source backbone Qwen2.5-VL-3B-R2R-low-level Vebbern (2025) and a closed-source model GPT-4 OpenAI (2023) to assess robustness across different model families.

Table 2: Results on VLN-CE-Isaac validation (H1 robot).

	Observation	VLN-CE-Isaac				
	S.RGB	NE↓	OS↑	SR↑	SPL↑	**RSR↑**
Navila	✓	5.86	54.6	45.3	40.3	-
GPT-4 (NavGPT)	✓	6.07	49.5	39.0	35.6	-
- Prompt	✓	6.02	52.2	42.4	38.7	18.3
- Graph-based	✓	5.73	55.8	47.9	40.3	38.1
- **RollbackNav**	✓	5.59	58.7	50.1	45.4	61.9
Qwen2.5-VL-3B	✓	7.81	30.4	22.8	21.5	-
- Prompt	✓	7.70	33.6	25.2	27.0	9.2
- Graph-based	✓	7.28	39.1	32.6	33.2	31.7
- **RollbackNav**	✓	6.11	49.7	45.3	39.5	54.4

Figure 4: An example in the Isaac-Sim.



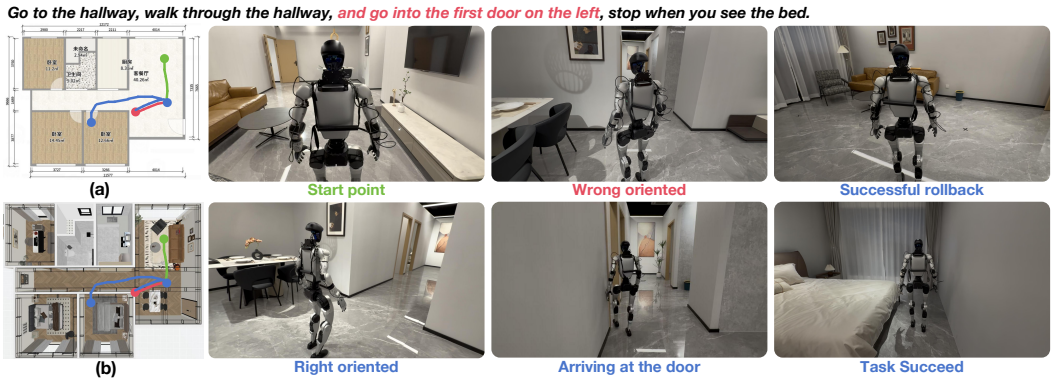


Figure 5: A real-world test case, the robot initially selected an insufficient turning angle, which resulted in an incorrect forward direction. Nevertheless, **RollbackNav** successfully rollback the robot to the right position and enabled it to complete the navigation task.

Evaluation Settings. To ensure fair comparison, we evaluate the following execution settings:

- **Baseline**: the model directly outputs low-level actions without explicit rollback mechanisms.
- **Prompt-based**: the model is prompted to reflect on execution errors and decide whether to rollback.
- **Graph-based**: the rollback strategy is based on a trajectory graph constructed during execution.
- **RollbackNav**: our program-guided approach with physically rollback and principled recovery.

Evaluation matrix. We adopt the VLN evaluation metrics, including NE↓, which measures the distance between the final agent position and the goal; OS↑, which indicates whether the agent reaches the success radius at any point along the trajectory; SR↑, the proportion of trajectories that successfully end within the goal threshold; and SPL↑, which evaluates success while penalizing inefficient path lengths, and introduce a new metric **RSR**↑ in Section 4.5 to quantify rollback reliability.

4.2 Results on R2R and RxR

We evaluate **RollbackNav** on the R2R Anderson et al. (2018) and RxR Ku et al. (2020), where agents follow language instructions to navigate in unseen environments. Results are shown in Table 1.

Although adding rollback can improve navigation, effectiveness depends on how rollback decisions are made. Prompt-based methods yield limited gains, as decisions rely solely on implicit reasoning without execution-level structure, performing similarly to the baseline on R2R and worse on RxR due to frequent incorrect rollbacks. Graph-based approaches achieve moderate improvements by introducing trajectory structure, but lack a principled rollback point selection mechanism and thus suffer from ambiguous failure localization, especially in long-horizon settings. In contrast, **RollbackNav** outperforms all baselines by selecting rollback points at the program level and explicitly verifying recovery correctness. By jointly modeling where to roll back and how rollback actions are executed, it enables more reliable recovery and improved performance across both benchmarks. Qualitative examples are shown in Figure 3.

Table 3: Real-world evaluation.

	Easy	Hard	Overall success rate
Number of total	400	100	500
GPT-4 (NavGPT)	67	37	61
- Prompt	70	40	64
- Graph-based	87	47	79
- **RollbackNav**	92	67	87

Table 4: **RollbackNav** as supervision for VLN training.

	R2R Val-Unseen				RxR Val-Unseen			
	NE↓	OS↑	SR↑	SPL↑	NE↓	OS↑	SR↑	SPL↑
Navid	5.47	49.1	37.4	35.9	8.41	34.5	23.8	21.2
Navid trained by **RollbackNav**	5.11	53.6	42.8	37.1	7.52	39.2	25.9	28.8

4.3 Results on VLN-CE-Isaac

We further evaluate on VLN-CE-Isaac Cheng et al. (2024) using an H1 robot, where physical dynamics, collisions, and actuation errors are explicitly modeled. Unlike static benchmarks, this setting exposes failures from imperfect motion execution, making executable rollback and accurate rollback point selection critical. As shown in Table 2, **RollbackNav** outperforms all baselines.

Prompt-based rollback is unstable due to inaccurate step-level judgments, while graph-based methods suffer from imprecise rollback point selection under physical constraints. In contrast, **RollbackNav** maintains robust performance by executing physically consistent rollback and resuming from program steps verified through perceptual consistency. Notably, the performance gap here is larger than on R2R and RxR, highlighting the importance of program-guided rollback under realistic dynamics and demonstrating its suitability for reset-free real-world deployment.

4.4 Real-World Experiments

We validate **RollbackNav** in real-world settings where rollback cannot rely on simulator resets. A Unitree G1 robot is deployed in a home-like indoor environment to execute 500 manually designed navigation instructions, including 400 easy tasks within a single room and 100 hard tasks requiring navigation across rooms, with success determined by human annotation of goal completion.

As shown in Table 3, **RollbackNav** achieves an overall success rate of 87%, outperforming graph-based rollback (79%) and prompt-based methods (64%). The advantage becomes more pronounced in long-horizon and multi-goal scenarios: on 100 cross-room tasks, **RollbackNav** reaches 67% accuracy, substantially higher than competing approaches. These results demonstrate the effectiveness of program-guided rollback for reset-free recovery and reliable real-world navigation.

4.5 Reliability Analysis via Recovery Success Rate

Standard metrics such as SR and SPL measure overall task performance but do not capture how effectively a system recovers from execution failures. To evaluate reliability from a recovery perspective, we introduce **Recovery Success Rate (RSR)**, defined as the proportion of rollback attempts that successfully recover from the corresponding execution failure:

$$\text{RSR} = N_{\text{successful recoveries}} / N_{\text{rollback attempts}}.$$

We evaluate RSR on R2R, RxR, and VLN-CE-Isaac under the same execution settings as in Section 4.1. For prompt-based and graph-based baselines, rollback attempts are counted whenever the method explicitly initiates backtracking. **RollbackNav** consistently achieves the highest RSR across all benchmarks, with the largest gains observed in VLN-CE-Isaac, where realistic dynamics frequently induce execution errors. While alternative methods sometimes improve SR, their lower RSR indicates that many rollback attempts fail to restore stable execution states. In contrast, the high RSR of **RollbackNav** demonstrates that its program-guided rollback more reliably re-establishes semantically and physically consistent intermediate states, validating its effectiveness for robust navigation under execution noise.

While **RollbackNav** achieves a high RSR, this metric mainly reflects recovery effectiveness after failures are detected. Overall system performance also depends on the accuracy of the semantic consistency-based detection stage that triggers rollback. Improving this detection component remains an important direction for future work.

Table 5: Ablation study on different program generators.

	R2R Val-Unseen				VLN-CE-Isaac			
	NE↓	OS↑	SR↑	SPL↑	NE↓	OS↑	SR↑	SPL↑
RollbackNav (GPT-4)	5.28	63.8	57.4	42.6	5.59	58.7	50.1	45.4
- by DeepSeek Coder (236B)	5.24	64.3	56.9	41.3	5.64	58.3	49.3	44.7
- by Code Llama (70B)	5.31	64.4	56.0	41.8	5.78	56.8	49.1	44.5

Table 6: Ablation study on rollback verification.

Verification	R2R	RxR	VLN-CE-Isaac
w/o Verification	37.4	26.3	35.9
RollbackNav	**57.9**	**39.1**	**49.7**
Oracle	59.8	43.6	52.1

4.6 ROLLBACKNAV IMPROVES VLN VIA SUPERVISION

Beyond improving execution-time robustness, we examine whether **RollbackNav** can provide effective supervision for training VLN models. We collect successful trajectories generated by **RollbackNav**, which include semantically consistent rollback anchors and corrected navigation paths, and use them as additional training data for a standard VLN agent (Navid). These trajectories expose the model to structured recovery behaviors absent from standard datasets.

Training Navid with the augmented data leads to consistent gains on R2R and RxR compared with training on original data. This result indicates that program-guided rollback supplies informative supervision on error-prone states and recovery strategies, demonstrating that **RollbackNav** not only improves execution robustness but also enhances the learning of VLN agents.

4.7 ABLATION STUDIES

We analyze key design choices of **RollbackNav** through ablations. We study program generation robustness and rollback verification in the main paper, with additional analyses of rollback behavior, action-space design, and verification granularity provided in Appendix A.

Program Generation Backbone. We first examine whether the effectiveness of **RollbackNav** depends on the choice of code-generating backbone. We compare GPT-based systems and other code-pretrained open-source models Roziere et al. (2023); Guo et al. (2024), while keeping the program specification and rollback mechanisms fixed. As shown in Table 5, different backbones achieve largely comparable performance across benchmarks, with only minor variations. While GPT-based models exhibit slightly stronger results in our main experiments, similar performance can be approached by scaling up open-source code-pretrained models. This suggests that the effectiveness of **RollbackNav** is not tied to proprietary model capabilities, but rather stems from the proposed programmatic formulation, which consistently elicits valid and executable programs across diverse backbones. Overall, these findings support the claim that **RollbackNav** is model-agnostic.

Effect of Rollback Verification. To evaluate the effectiveness of rollback verification, we compare three settings: *(i) w/o Verification*, which performs rollback solely through inverse actions without subsequent visual verification; *(ii)* **RollbackNav**(Qwen), which verifies the recovered state using feature-level matching and performs corrective adjustments when necessary; and *(iii) Oracle Verification*, which uses ground-truth coordinates for recovery verification and correction. We report Overall Success (OS) on R2R, RxR, and VLN-CE-Isaac, as summarized in Table 6. Without verification, the OS is only 37.4, 26.3, and 35.9, respectively, whereas our feature-based verification substantially improves the OS to 57.9, 39.1, and 49.7. These results indicate that inverse actions alone are insufficient to reliably recover from execution errors due to accumulated deviations and imperfect physical reversibility. Moreover, our feature-based verification approaches the oracle setting, which achieves 59.8, 43.6, and 52.1 OS, demonstrating that visual feature matching provides an effective recovery verification mechanism without relying on privileged ground-truth state information.

5 CONCLUSION

We presented **RollbackNav**, a program-guided navigation paradigm that treats rollback as a first-class, physically executable operation for reliable embodied navigation. By representing VLN behaviors as executable programs with intermediate states, **RollbackNav** enables step-level error localization, navigation-consistent rollback actions, and recovery verification via perceptual feature consistency. Extensive results on VLN benchmarks, high-fidelity simulation, and real-world robotic deployments show consistent improvements over heuristic rollback baselines.

AI DISCLOSURE

In this work, we used generative AI tools to edit and polish the manuscript, including improving its grammar, clarity, and readability. We also used generative AI tools to assist with code development. All AI-assisted textual revisions were reviewed by the authors to ensure that they preserved the intended meaning and accurately reflected the research methods and results. AI-assisted code was reviewed and verified for correctness by multiple authors. We take full responsibility for the final content of this work.

ETHICS STATEMENT

This work uses publicly available datasets and open-weight models and does not collect private or sensitive personal information. Human review is limited to validating the semantic consistency and correctness of controlled queries. We acknowledge potential biases in the source datasets, evaluated models, and automated evaluators.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton Van Den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3674–3683, 2018.

An-Chieh Cheng, Yandong Ji, Zhaojing Yang, Zaitian Gongye, Xueyan Zou, Jan Kautz, Erdem Bıyık, Hongxu Yin, Sifei Liu, and Xiaolong Wang. Navila: Legged robot vision-language-action model for navigation. *arXiv preprint arXiv:2412.04453*, 2024.

Hongyu Ding, Ziming Xu, Yudong Fang, You Wu, Zixuan Chen, Jieqi Shi, Jing Huo, Yifan Zhang, and Yang Gao. Lavira: Language-vision-robot actions translation for zero-shot vision language navigation in continuous environments. *arXiv preprint arXiv:2510.19655*, 2025.

Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, et al. Palm-e: An embodied multimodal language model. *openreview.net*, 2023.

Jiafei Duan, Samson Yu, Hui Li Tan, Hongyuan Zhu, and Cheston Tan. A survey of embodied ai: From simulators to research tasks. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6(2):230–244, 2022.

Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, et al. Foundation models in robotics: Applications, challenges, and the future. *The International Journal of Robotics Research*, 44(5):701–739, 2025.

Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. Speaker-follower models for vision-and-language navigation. *Advances in neural information processing systems*, 31, 2018.

Pascale Fung, Yoram Bachrach, Asli Celikyilmaz, Kamalika Chaudhuri, Delong Chen, Willy Chung, Emmanuel Dupoux, Hongyu Gong, Hervé Jégou, Alessandro Lazaric, et al. Embodied ai agents: Modeling the world. *arXiv preprint arXiv:2506.22355*, 2025.

Weifan Guan, Qinghao Hu, Aosheng Li, and Jian Cheng. Efficient vision-language-action models for embodied manipulation: A systematic survey. *arXiv preprint arXiv:2510.17111*, 2025.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14953–14962, 2023.

Keji He, Kehan Chen, Jiawang Bai, Yan Huang, Qi Wu, Shu-Tao Xia, and Liang Wang. Everyday object meets vision-and-language navigation agent via backdoor. *Advances in Neural Information Processing Systems*, 37:49684–49705, 2024.

Siyuan Huang, Zhengkai Jiang, Hao Dong, Yu Qiao, Peng Gao, and Hongsheng Li. Instruct2act: Mapping multi-modality instructions to robotic actions with large language model. *arXiv preprint arXiv:2305.11176*, 2023.

Liyiming Ke, Xiujun Li, Yonatan Bisk, Ari Holtzman, Zhe Gan, Jingjing Liu, Jianfeng Gao, Yejin Choi, and Siddhartha Srinivasa. Tactical rewind: Self-correction via backtracking in vision-and-language navigation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6741–6749, 2019.

Alexander Ku, Peter Anderson, Roma Patel, Eugene Ie, and Jason Baldridge. Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4392–4412, 2020.

Xinqing Li, Xin He, Le Zhang, Min Wu, Xiaoli Li, and Yun Liu. A comprehensive survey on world models for embodied ai. *arXiv preprint arXiv:2510.16732*, 2025.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

Rui Liu, Wenguan Wang, and Yi Yang. Vision-language navigation with energy-based policy. *Advances in Neural Information Processing Systems*, 37:108208–108230, 2024.

OpenAI. Gpt-4 technical report, March 2023. URL `https://arxiv.org/abs/2303.08774`. arXiv:2303.08774.

Akhil Perincherry, Jacob Krantz, and Stefan Lee. Do visual imaginations improve vision-and-language navigation agents? In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 3846–3855, 2025.

Yanyuan Qiao, Wenqi Lyu, Hui Wang, Zixu Wang, Zerui Li, Yuan Zhang, Mingkui Tan, and Qi Wu. Open-nav: Exploring zero-shot vision-and-language navigation in continuous environment with open-source llms. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6710–6717. IEEE, 2025.

Wenda Qin, Andrea Burns, Bryan A Plummer, and Margrit Betke. Walk and read less: Improving the efficiency of vision-and-language navigation via tuning-free multimodal token pruning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 23567–23581, 2025.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Sahar Salimpour, Lei Fu, Kajetan Rachwał, Pascal Bertrand, Kevin O'Sullivan, Robert Jakob, Farhad Keramat, Leonardo Militano, Giovanni Toffetti, Harry Edelman, et al. Towards embodied agentic ai: Review and classification of llm-and vlm-driven robot autonomy and interaction. *arXiv preprint arXiv:2508.05294*, 2025.

Xiangyu Shi, Zerui Li, Wenqi Lyu, Jiatong Xia, Feras Dayoub, Yanyuan Qiao, and Qi Wu. Smartway: Enhanced waypoint prediction and backtracking for zero-shot vision-and-language navigation. *arXiv preprint arXiv:2503.10069*, 2025.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: program generation for situated robot task planning using large language models. *Autonomous Robots*, 47(8):999–1012, 2023.

Xinshuai Song, Weixing Chen, Yang Liu, Weikai Chen, Guanbin Li, and Liang Lin. Towards long-horizon vision-language navigation: Platform, benchmark and method. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 12078–12088, 2025.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 11888–11898, 2023.

Vebbern. Vebbern/qwen2.5-vl-3b-r2r-low-level: A low-level action space vision-and-language navigation model. `https://huggingface.co/Vebbern/Qwen2.5-VL-3B-R2R-low-level`, 2025. Accessed: 2026-01-22.

Liuyi Wang, Zongtao He, Ronghao Dang, Mengjiao Shen, Chengju Liu, and Qijun Chen. Vision-and-language navigation via causal learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 13139–13150, 2024.

Zhengcheng Wang, Zichuan Lin, Yijun Yang, Haobo Fu, and Deheng Ye. Seenav-agent: Enhancing vision-language navigation with visual prompt and step-level policy optimization. *arXiv preprint arXiv:2512.02631*, 2025.

Zun Wang, Jialu Li, Yicong Hong, Yi Wang, Qi Wu, Mohit Bansal, Stephen Gould, Hao Tan, and Yu Qiao. Scaling data generation in vision-and-language navigation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12009–12020, 2023.

Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*, 2025.

Junrong Yue, Yifan Zhang, Chuan Qin, Bo Li, Xiaomin Lie, Xinlei Yu, Wenxin Zhang, and Zhendong Zhao. Think hierarchically, act dynamically: Hierarchical multi-modal fusion and reasoning for vision-and-language navigation. *arXiv preprint arXiv:2504.16516*, 2025.

Siqi Zhang, Yanyuan Qiao, Qunbo Wang, Zike Yan, Qi Wu, Zhihua Wei, and Jing Liu. Cosmo: Combination of selective memorization for low-cost vision-and-language navigation. *arXiv preprint arXiv:2503.24065*, 2025.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. *arXiv preprint arXiv:2306.07863*, 2023.

A Additional Experimental Results

This section provides additional experimental results that further characterize the behavior and design choices of **RollbackNav**. While the main paper focuses on the overall navigation performance and the effectiveness of rollback verification, we provide supplementary analyses of rollback behavior, action-space design, and verification granularity. These experiments offer additional insights into how **RollbackNav** performs recovery under different execution settings and how its individual design choices affect navigation reliability.

A.1 Rollback Behavior and Recovery Statistics

To further characterize the execution behavior of **RollbackNav**, we analyze how frequently rollback is triggered and how reliably the agent recovers from execution failures. Specifically, we report the rollback ratio, the average number of rollback attempts per triggered episode, and the Rollback Success Rate (RSR), together with the Overall Success (OS). The results are reported for both GPT-4 and Qwen2.5-VL across R2R, RxR, and VLN-CE-Isaac.

Table 7: Rollback frequency and recovery statistics across different datasets and program generators.

Model	Dataset	#Samples	Rollback Ratio	Avg. Rollbacks	RSR ↑	OS ↑
GPT-4	R2R	1839	22.5%	1.3	88.9	63.8
	RxR	11006	16.9%	1.8	82.4	49.6
	VLN-CE	1077	10.1%	1.7	61.9	58.7
Qwen2.5-VL	R2R	1839	26.5%	1.5	76.4	57.9
	RxR	11006	19.8%	2.1	71.7	39.1
	VLN-CE	1077	17.2%	2.3	54.4	49.7

Across the three benchmarks, rollback is triggered in 10.1%–26.5% of the evaluated episodes, with an average of 1.3–2.3 rollback attempts per triggered episode. Despite these repeated recovery attempts, **RollbackNav** achieves a relatively high RSR across all settings, indicating that most triggered failures can be successfully recovered. The results also show consistent recovery behavior across different program generators, demonstrating that the rollback mechanism is not specific to a particular backbone.

A.2 Action Space Analysis

We further investigate the effect of the action space on the effectiveness of **RollbackNav**. We first compare different levels of action interfaces, including action-only APIs, extended APIs, and our program-based rollback framework. We then examine whether rollback remains effective when additional motion primitives are introduced into the underlying action space.

Effect of Action Interfaces. We compare three execution settings: *(i) Action-only APIs*, where the model directly outputs low-level navigation actions; *(ii) Extended APIs*, which provide additional execution-level interfaces; and *(iii) RollbackNav*, which augments programmatic execution with physically executable rollback and recovery. On the R2R benchmark, the corresponding OS scores are 27.1, 58.4, and 57.9, respectively. These results show that providing richer execution interfaces substantially improves navigation performance compared with directly predicting low-level actions, while our rollback mechanism achieves comparable performance without relying solely on an expanded action interface.

Rollback Space Design. We further evaluate **RollbackNav** under different primitive action spaces. Starting from the original four-action space, which includes forward motion, left/right rotation, and stop, we consider two extensions: adding a backward motion primitive and further augmenting the action space with left-right translational movements. As shown in Table 8, rollback consistently improves navigation performance across all three action spaces. With the original four actions, rollback improves SPL from 29.0 to 42.6 and achieves an RSR of 88.9. Adding backward motion yields an SPL improvement from 27.8 to 40.7 with an RSR of 87.4, while further introducing

Table 8: Effect of different action spaces on R2R. We report SPL and RSR with and without rollback.

Action Space	Rollback	SPL ↑	RSR ↑
Original Four Actions	w/o	29.0	–
	w/	42.6	88.9
+ Backward	w/o	27.8	–
	w/	40.7	87.4
+ Left-right Translation	w/o	32.3	–
	w/	46.9	89.4

Table 9: Effect of verification granularity on Overall Success (OS).

Verification Strategy	R2R	RxR	VLN-CE-Isaac
w/o Verification	37.4	26.3	35.9
Full Completion	57.9	39.1	49.7
Each Action	58.4	39.3	50.2

translational movements improves SPL from 32.3 to 46.9 and achieves an RSR of 89.4. These results demonstrate that the effectiveness of program-based rollback is robust to variations in the underlying action space, rather than being tied to a particular set of motion primitives.

A.3 Verification Granularity

We further investigate the granularity of rollback verification by comparing three settings: *(i) w/o Verification*, where rollback is executed without subsequent visual verification; *(ii) Full Completion*, where verification is performed after the entire rollback sequence is executed; and *(iii) Each Action*, where the recovered state is verified after every rollback action. The results are summarized in Table 9.

The results show that verifying after each individual rollback action provides only marginal improvements over verification after the complete rollback sequence, with gains of 0.5, 0.2, and 0.5 OS points on R2R, RxR, and VLN-CE-Isaac, respectively. This suggests that full-completion verification provides a favorable balance between recovery reliability and verification overhead, while fine-grained verification after every action offers limited additional benefit.

B APPENDIX OF ROLLBACKNAV: PROGRAM-BASED ROLLBACK FOR RELIABLE EMBODIED NAVIGATION.

Program generation prompt for VLN execution

Write a Python function named `execute_command()` that can be executed to navigate a robot according to the given VLN instruction.
This is program generation. Your output must be executable Python code.
The function should implement a *multi-step* navigation policy: during execution, it repeatedly observes the environment, branches on perception results, and returns an `Action` to be executed; if the instruction is satisfied, it returns `Action(STOP)`.
You MUST keep using the provided ViperGPT visual tools for perception and grounding (do not replace them):
- `img = ImagePatch(take_photo())`
- `ImagePatch.find()`
- `ImagePatch.exists()`
- `ImagePatch.simple_query()`
- `ImagePatch.verify_property()`
- `ImagePatch.best_text_match()`
- `distance()`
- `bool_to_yesno()`

Robot actions (must be returned as an `Action`; only these are allowed):
- `Action(FORWARD, steps=...)` or `Action(FORWARD, distance=...)`
- `Action(TURN_LEFT, degrees=...)`
- `Action(TURN_RIGHT, degrees=...)`
- `Action(STOP)`

Other tool:
- `get_pose()`
- `angle_to_face(img)`
- `distance_estimation(img)`
- `error_localization(action_history)`
- `verify_state(anchor)`

Program requirements:
- The program must take a new photo when needed and use perception to decide which branch to execute.
- The program may contain multiple branches; each branch should return a single `Action` for the next step.
- Use simple spatial reasoning with Python, e.g., left/right via `h_center`, up/down via `v_center`, nearest via `distance()`.
- If the instruction is satisfied, return `Action(STOP)`.

Style constraints (very important):
- Output only Python code. Do not output explanations, dictionaries, or extra text.
- Define exactly one top-level function `execute_command()`.
- The function must follow the pattern `img = ImagePatch(take_photo())` and then `if`/`else` branching that ends with `return Action(...)`.

VLN instruction: INSERT_INSTRUCTION_HERE