RoboState: State-Aware Planning for Continual Embodied Instruction Following in Open Dynamic Environment

Anonymous authors

Abstract

Continual embodied instruction following (CEIF) is prevalent in long-term human-agent cohabitation tasks such as home service, elderly care, and autonomous inspection. CEIF is particularly challenging in open dynamic environments which are affected by human activity between consecutive observations, invalidating the agent's previous recognition of environment state. Existing methods make decisions based on surface-level environmental information, which causes planning to depend on stale or misaligned information and involve unavailable objects, ultimately leading to failures.

We propose RoboState, a state-aware framework for CEIF in open dynamic environments, which operates through a three-stage pipeline of (1) parsing an instruction into goal states, (2) aligning and tracking the current states of the relevant objects, and (3) constructing an executable plan from the current states to the goal states. We evaluate RoboState on a new benchmark containing 120 CEIF instructions as well as on the existing ExRAP benchmark. The results show that RoboState remains robust across different dynamic difficulty levels, instruction scales, instruction types, and LLM backbones, and achieves higher task success rates compared with the state-of-the-art (SOTA) methods.

1 Introduction

There is a growing demand for embodied agents to handle continual instructions in long-term human-agent cohabitation tasks such as home service, elderly care, and autonomous inspection (Lyu et al., 2026; Chang et al., 2025). This task is referred to as continual embodied instruction following (CEIF), which requires agents to continuously monitor the environment, determine when continual instruction is triggered, and execute the task for the satisfied condition.

This requirement becomes particularly challenging in open dynamic environments, where relevant objects may be unknown in advance and human activity inevitably introduces gaps and discontinuities between consecutive observations (Wu et al., 2025). For example, for the instruction "keep me cool when I am in the bedroom in the evening," the agent operating in the bedroom may need to adjust the fan and the air conditioner (AC) and prepare a cup of ice water. Outside the agent's observation, a family member may replace the clean cup in bedroom with a previously used living-room cup containing residue, causing agent to mistake the nearby cup for the original one. These gaps and discontinuities can cause the agent's knowledge to diverge from the actual environment state, which undermines three aspects: the realizability of goal states (e.g., the originally selected cup is no longer available), the accuracy of object states (e.g., a replacement cup is mistaken for the original), and the executability of state transitions (e.g., the plan starts from a wrong state of the cup). Hence, reliable CEIF in open dynamic environments requires realigning goal states, object states, and plans with the environmental evidence following human-induced changes.

Despite its practical significance, research on CEIF of open dynamic environments remains very limited. ExRAP (Yoo et al., 2024) is the sole study specifically designed for CEIF. However, ExRAP assumes a closed environment that changes only through the agent's

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.

own actions, and ignores the divergence between its memory and environment, causing task failure. Other notable efforts (Singh et al., 2022; Song et al., 2023; Ahn et al., 2022; Huang et al., 2022a) are broader embodied instruction following studies instead of CEIF. Most of them process an instruction once upon receipt for execution, without providing mechanism for repeatedly determining whether a triggering condition holds. Consequently, they cannot directly handle continual tasks. Fundamentally, existing studies lack explicit checks of consistency between goals, remembered object states, and current observations, and may fail to detect or correct the divergence caused by open dynamic environment. For example, interpreting instructions as fixed action sequences can leave the intended goals unrealizable. Updating memory without checking consistency across observations can conflate distinct object instances. Generating plans without checking object availability can produce plans that refer to unavailable objects or unsupported skills.

Our key insight is to explicitly represent states so that discrepancies between the agent's knowledge and the environment can be modeled and corrected. Accordingly, we represent the instruction, the environment, and the plan as goal states, object state trajectories, and state-transition paths, respectively. These representations allow us to identify inconsistent states and update them, using verified observational evidence.

We propose RoboState, a state-aware CEIF framework with a pipeline of goal reasoning, state-consistent memory, and state-transition-graph (STG) based planning. These modules respectively parse an instruction into goal states, align the current states of the relevant objects, and construct an executable plan from the current states to the goal states once the instruction's triggering condition is satisfied. Together, they maintain consistency between goal states, object states, plans and environment changes. In goal reasoning, RoboState generates intended goal states, then performs intention-environment reconciliation to check each goal's realizability against the current memory and seek alternatives for invalid goals (See Section 3.2). In state-consistent memory, RoboState organizes environmental knowledge in an object-centric manner, and assigns each new observation to its corresponding object via state alignment, thereby maintaining reliable current states for planning (See Section 3.3). In STG-based planning, RoboState constructs the STG for the relevant objects, and then derives a unified executable plan via path search and merging (See Section 3.4).

To evaluate RoboState, we construct the first dedicated benchmark for this setting, with 120 continual instructions across VirtualHome and ALFRED. The size is larger than those used in SOTA studies (35 in ExRAP). We additionally evaluate RoboState on the ExRAP benchmark to compare it with SOTA methods under ExRAP's settings. Across both benchmarks, RoboState achieves higher task success rates than the representative and SOTA methods. And its advantage becomes more pronounced under higher dynamic difficulty, larger instruction scales, and vague instruction expressions. Ablation studies further demonstrate the impact of each module and its core components.

Contributions. This paper makes the contributions: First, we propose a novel state-aware planning framework to address CEIF in open dynamic environments. We also demonstrate the effectiveness and robustness of RoboState through extensive experiments in the VirtualHome and ALFRED environments. Finally, we construct the first dataset for CEIF in open dynamic environments, and will release our artifacts and datasets.

2 Related Work

Continual embodied instruction following. Research on CEIF in open dynamic environments remains limited. ExRAP (Yoo et al., 2024) is the only study explicitly developed for CEIF. Since ExRAP assumes a closed environment which does not hold in real human-involved open dynamic environments, ExRAP's memory may then associate reappearing objects with incorrect prior instances, while generated plans may reference unavailable objects, causing task failures and execution loss. Other related efforts focus on broader embodied instruction following in three directions. The first maps natural-language instructions into executable actions (Huang et al., 2022a; Ahn et al., 2022; Song et al., 2023; Singh et al., 2022; Liang et al., 2023). The second augments planning with memory or environment representations (Kim et al., 2023; Chen et al., 2023; Yang et al., 2024; Rana et al., 2023;

Sarch et al., 2023). The third addresses adaptation and errors during planning (Wang et al., 2025; 2023; Xu et al., 2026; Huang et al., 2022b). These methods treat each instruction as a one-shot task whose applicability is evaluated once before execution. They therefore cannot repeatedly reevaluate trigger conditions while an instruction remains active, limiting their applicability to CEIF.

Furthermore, existing approaches have limited effectiveness, due to relying on surface-level representations instead of underlying factors. Specifically, they interpret instructions as literal actions instead of the intended goal states, construct memory from the isolated observations rather than the environmental changes across observations, and generate plans directly by the model rather than deriving them from feasible state transitions. RoboState addresses this limitation in a state-aware manner, which maps instructions into intended goal states, aligns and tracks the current states of relevant objects, and generates feasible plans by searching paths from the current states to the goal states over STG.

Embodied agents for open dynamic environment. Research on embodied agents for open dynamic environments remains limited, and existing methods lack the support for CEIF planning for two limitations. First, they address adjacent problems rather than CEIF planning. For example, SGM (Kurenkov et al., 2023) estimates object locations rather than executable plans. KnowNo (Ren et al., 2023) determines when to seek assistance. And InternVLA (Chen et al., 2025) handles single-skill control rather than long-horizon planning. Second, these methods model environmental dynamics in isolated facets rather than within a unified framework. SGM and LookPlanGraph (Onishchenko et al., 2025) model changing object locations. KnowNo estimates planner uncertainty for assistance request. FLARE (Kim et al., 2025) addresses plan-state discrepancies. InternVLA improves control robustness under environmental changes. Omitting any of these aspects (e.g., location, availability, identity, or existence status) can cause subsequent triggers to be evaluated or plans to be executed on false premises, leading to task failures.

We characterize the impact of open dynamics as uncontrolled divergence between the agent's knowledge and the true environment state. Accordingly, RoboState maintains an object-centric memory that tracks each object's state over time and applies a unified state-alignment mechanism to detect and correct this divergence. This keeps the agent's knowledge consistent with the environment.

3 Approach

3.1 Overview

RoboState takes a continual instruction I as input and outputs an executable plan π through a pipeline that decouples *what the user wants*, *what the world currently is*, and *how to act*.

Hierarchical Reasoning for Goal States. Given the instruction I, this module derives initial goal states (G, P), where G specifies the actionable objects and associated operations. P contains the key parameters of triggering condition and these objects. Then, RoboState adapts goal states to human-induced changes via intention-environment reconciliation. Firstly, reconciliation analyzes realizability of each goal state. For each unsupported goal state, RoboState searches the current memory for an alternative satisfying the original requirements while preserving G. Finally, RoboState requests option-based user clarification for resolved changes and ensures the consistency between goal states and environment.

State-Consistent Object Memory. RoboState represents the open dynamic environment as an object-centric knowledge graph $\mathcal{G}^{(k)}$. Each task-relevant object is assigned a persistent slot, which is characterized by state-attribute, temporal, spatial, and rule relations. Observations are processed accurately into memory through three stages: (1) prediction stage derives expected state sets from each slot's recent confirmed states using skill preconditions and effects, providing historical context for evaluating new observations, (2) candidate generation utilizes an LLM to propose candidate slots and provide the related evidence, (3) state alignment associates observed objects with persistent slots by considering state-attribute, temporal, and spatial continuity together with rule consistency.

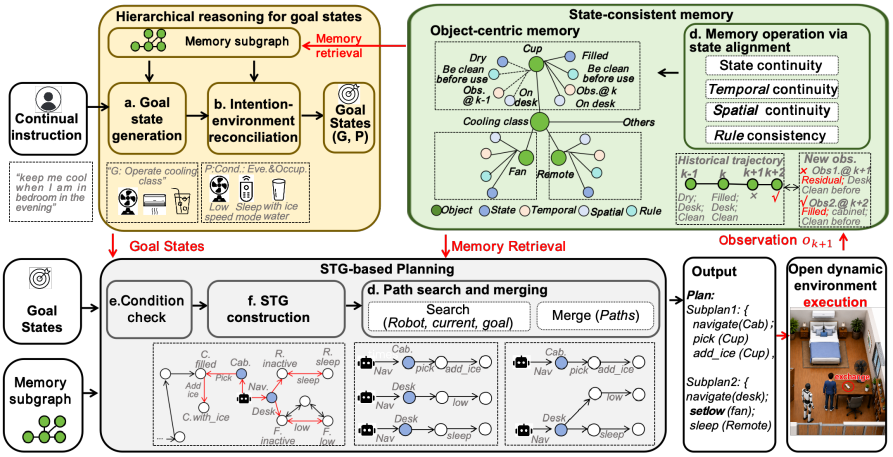

Figure 1: Overview of RoboState.

STG-Based Task Planning. Given (G, P) and $\mathcal{G}^{(k)}$, RoboState identifies the task-relevant objects, retrieves their corresponding memory subgraph, and evaluates the triggering condition specified by G. Once the condition is satisfied, RoboState constructs the STG for these objects, in which nodes represent object states and edges represent feasible skill-induced transitions. Path search then finds paths from each object's current state to its goal state over STG and composes the resulting paths into executable plan π.

3.2 Hierarchical Reasoning for Goal States

Executing the literal actions for continual instruction directly is unreliable, because the same instruction may require different or even opposite actions under different environment states. Therefore, RoboState maps I into structured goal states (G, P), where G is a parameterized goal predicate and P contains its concrete parameter values. Together, they describe the environment states intended to be maintained. Formally,

$$(G, P) = \mathcal{R}(I, \mathcal{G}^{(k)}), \tag{1}$$

where the mapping $\mathcal{R}$ first generates an initial goal state and then revises its bindings in response to environmental changes through intention–environment reconciliation.

Initial goal state generation. For continual instruction, its literal action describes a surface means to an underlying goal state. To recover this goal state, we model the inference from the surface action to the actionable root intention as a chain of successive *why* questions on LLM. Inspired by the widely-used Five Whys root-cause analysis (Ohno, 2019), RoboState utilizes a Five-Whys prompt on LLM for this step, denoted as $G = \phi_{why}(I)$, where G is a goal predicate that specifies the actionable objects and operations on them required to achieve the goal. Then, RoboState binds parameters of goal predicate G to concrete values, denoted by $P = \phi_{acq}(G, \mathcal{G}^{(k)})$. For each parameter in P, ϕ_{acq} infers its goal value from the experience and knowledge stored in the memory subgraph, such as applicable rules, and experienced goal states observed in similar situations. ϕ_{acq} retrieves candidate values for each parameter from the relevant memory subgraph by cosine similarity and requests option-based user clarification only when the evidence is insufficient.

Intention-environment reconciliation. An initially valid goal state (G, P) may become inconsistent with the environment after human intervention, for example, when a bound cup is moved out of the required room. Reconciliation records the memory facts that support each binding, rechecks the binding when these facts change, and repairs it only if it is no longer supported by the current memory. Formally, using the recorded supporting facts and the current memory, ϕ_{robust} revises P while preserving G,

$$(G', P') = \phi_{\text{robust}}\big(G, P, \mathcal{G}^{(k)}\big). \tag{2}$$

where ϕ_{robust} checks the current memory to determine whether each bound object exists and satisfies its availability and location requirements, whether the triggering conditions can be evaluated, and whether the target values are consistent with known constraints. It records the supporting facts used in these checks.

After each memory update (Section 3.3), ϕ_{robust} identifies changed supporting facts and rechecks the affected bindings. Bindings whose requirements remain satisfied are retained. For each binding that is unsupported, ϕ_{robust} searches the current memory for an available alternative that satisfies the original requirements, without changing G. Finally, ϕ_{robust} checks the revised goal state (G, P') for joint consistency. It forwards the goal state to planning (Section 3.4) only when all required bindings are resolved and the consistency check passes. The prompt for ϕ_{why} and the pseudocode for ϕ_{robust} are available in repository.

In Fig. 1, the generated goal state sets the location to `bedroom`, the triggering conditions to `occupied` and `evening`, and the target states of the fan, remote, and cup to `low speed`, `sleep mode`, and `with_ice_water`, respectively. The cup binding is supported by facts indicating that the selected cup exists, is available, and is located in the bedroom. Human intervention swaps the cups between the two rooms, invalidating its recorded location fact. ϕ_{robust} rebinds the cup to an available alternative or, if none is available, reports that the binding is unrealizable, while retaining the location, trigger, fan, and remote bindings.

3.3 State-Consistent Object-Centric Memory

RoboState utilizes an object-centric memory with three-stage alignment procedure of prediction, candidate generation and state alignment, to bridge the gaps and discontinuities between observations through state alignment.

Object-centric memory. Fig. 1 shows the architecture. After k updates, the memory is denoted as $\mathcal{G}^{(k)} = (V^{(k)}, E^{(k)})$, an object-centric graph containing object, state-attribute, temporal, spatial, and rule nodes. Each edge has a relation type and a validity interval $[k_{from}, k_{to})$, where $k_{to} = \infty$ denotes an active relation and k indexes memory updates rather than actual time. Every task-relevant object o_i is represented by a persistent object node v_i^{obj}. And object slot $S_i^{(k)}$ is the ego-subgraph centered at v_i^{obj} at version k, which comprises the four relation categories,

$$S_i^{(k)} = \big(R_{i,state}^{(k)},\, R_{i,temp}^{(k)},\, R_{i,spat}^{(k)},\, R_{i,rule}^{(k)}\big), \tag{3}$$

where $R_{i,state}^{(k)}$ represents the object's state, whereas $R_{i,spat}^{(k)}$, $R_{i,rule}^{(k)}$ record its observation location, and applicable rules, respectively. Further, $R_{i,temp}^{(k)}$ records observation at version k and the most recent skill $a^{(k-1)}$ applied to the object for alignment.

RoboState performs memory consolidation and retrieval similarly to ExRAP (Yoo et al., 2024), we focus on incorporating new observations into memory through the state-alignment.

Memory Operation based on state alignment *Step-1. State prediction.* Each skill in the skill set $\mathcal{A}$ specifies its preconditions and effects. Given a confirmed object state, the effects of all skills whose preconditions hold in that stateare regarded as its expected states. For example, if the preconditions of `add_ice` hold for a cup in state `filled`, applying its effects yields a candidate state labeled as `with_ice`. RoboState constructs an expected state set for each of a slot's last n confirmed states. When missing relations prevent determining whether a skill's preconditions hold, the corresponding expected successor state is marked as unknown. The source state is also retained as a no-change hypothesis.

Step-2. LLM-based candidate generation. Each new observation j is parsed into a snapshot $\hat{S}_j^{(k)}$ containing the same four relation types as an object slot. An LLM compares the snapshot with the recent histories of slots with the same category or similar names, proposes candidate slots, and labels relations supporting or contradicting each candidate. RoboState checks each label against the observation or memory and discards labels without substantiation. The candidate set C_j combines all slots with LLM-proposed candidates. The candidate set bounds the assignment search, while the below identity alignment provides supporting evidence for acceptance.

Step-3. State alignment. At update k, a new observation is parsed into a snapshot $\hat{S}^{(k)}$ with the same four components. RoboState then aligns it with a slot in the candidate set C, creates a new slot, or refuses the update when the evidence is ambiguous. We formulate this process as state-trajectory alignment, which selects the slot whose recent trajectory most plausibly explains $\hat{S}^{(k)}$ as its next observation. For each candidate slot i, the alignment score is

$$\mathrm{A}_i\big(\hat{S}^{(k)}, H_i^{(k,n)}\big) = S_{id} + \sum_{c \in C_{cat}} S_c + S_{dyn}, \tag{4}$$

where identity term S_{id} measures name and category agreement. Each relation-consistency term S_c ($c \in C_{cat} = \{attr, temp, spat, rule\}$) compares the corresponding elements of $\hat{S}^{(k)}$ with those of the candidate slot. S_{id} and S_c are computed as cosine similarities between word-count vectors. Prediction consistency S_{dyn} equals 1 if the predicted state from $S_i^{(m)}$ is consistent with $\hat{S}_j^{(k)}$ and 0 otherwise.

We formulate the association between the multiple observations in an update and the existing object slots as a global matching problem. Let A_{ji} be the reward for matching observation j to slot i. RoboState utilizes existing strategy (Ratnayaka, 2024) to solve a linear assignment that maximizes total reward, subject to each observation matching at most one slot and each slot receiving at most one observation.

In Fig. 1, the cup originally in the bedroom has a confirmed state of *filled*. After the cups are swapped between the two rooms, a cup in state *residual* is observed in the bedroom. Because the observed *residual* state is inconsistent with all known expected states for the original bedroom cup's slot, this pair receives $S_{\mathrm{dyn}} = 0$. This prevents the livingroom cup from being incorrectly merged into the bedroom cup's slot and prompts the agent to search elsewhere. A later compatible observation, such as a valid filled state, can be assigned to the bedroom cup's slot (Fig. 1.d).

3.4 State-transition-graph-based Planning

Goal state (G, P) can be achieved by multiple skill sequences, and their feasibility depends on the currently-available objects, their states and locations, and the robot-supported skill set K. An LLM-generated end-to-end skill sequence may violate these constraints by referencing unsupported skills or objects unavailable in the current environment. Therefore, RoboState restricts the LLM to parsing the goal state and the retrieved memory subgraph into structured planning inputs. The final action sequence is derived through path search over an explicit state-transition graph (STG), each edge of which is grounded in a robot-supported skill. Given (G, P) and $\mathcal{G}^{(k)}$, the executable plan is given by

$$\pi = \mathcal{T}\big(G, P, \mathcal{G}^{(k)}\big), \tag{5}$$

where the mapping $\mathcal{T}$ is realized in the following four steps.

Condition Check and State Parse. RoboState retrieves the subgraph relevant to (G, P) from $\mathcal{G}^{(k)}$, identifies the corresponding object set O, and extracts each object's current and goal states. RoboState then uses these current states to evaluate the triggering condition. Once triggered, this step parses state transitions (s_i, a, s_j) from the memory $\mathcal{G}^{(k)}$ and the skill set K, which consist of a source state s_i, a target state s_j, and the skill a for the transition. For $\mathcal{G}^{(k)}$, two consecutive states in an object's trajectory $H_i^{(k,n)}$ provide (s_i, s_j), while the recorded skill provides the edge a. If goal-relevant state changes are not in memory, we instantiate them from the preconditions and effects of skill K.

State Transition Graph Construction. This step constructs a state-transition graph $\mathcal{M}^{stg} = (V, E)$. For every grounded transition (s_i, a, s_j), the graph contains the corresponding object-specific state nodes and an edge from s_i to s_j labeled with the skill a. To include object access process, node set V is extended with the robot node representing its current location, location nodes of objects. And E additionally contains navigation edges connecting the robot to the locations and association edges connecting current states to the locations of the objects. Every skill-transition edge or navigation edge corresponds to a robot-supported skill, whereas association edges are used as search constraints.

Path Search and Merging. On $\mathcal{M}^{stg}$, we search for a path for each relevant object o

$$\pi_o = \text{GraphSearch}\big(\text{Robot},\, s_o^{cur},\, s_o^{goal}\big), \tag{6}$$

which starts from the robot node, reaches the object location along navigation edges, and then drives the object from its current state s_o^{cur} to its goal state s_o^{goal} along skill-transition edges. The action-labeled edges on the resulting path form an executable skill sequence.

The Merge operator reorders operations of per-object paths and removes redundant ones.

$$\pi = \text{Merge}\big(\{\pi_o\}_{o\in O};\, C_{order} \cup C_{resource} \cup C_{spatial}\big), \tag{7}$$

RoboState reorders operations based on precedence dependencies C_{order}, serializes actions competing for the same exclusive resource via resource constraint $C_{resource}$, and merges navigation segments by spatial constraint $C_{spatial}$ with a shared destination to avoid repeated visits. The details and pseudocode are available in repository.

In Fig. 1, RoboState first retrieves the current states of cup, fan, AC, and its remote from memory, and verifies the occupancy-and-time trigger. Then, RoboState constructs the STG, and searches for paths for objects, (*cup: filled → with_ ice* via `pick` and `add_ice`, *fan: inactive → low* via `setLow`, and *AC: inactive → sleep* via `setSleep`). RoboState merges paths into final plans: {`navigate`(*Cabinet*); `pick`(*Cup*); `add_ice`(*Cup*)} and {`navigate`(*desk*); `low`(*fan*); `sleep`(*remote*)} (Fig. 1.g).

4 Experiment

We evaluate RoboState in **VirtualHome** (Puig et al., 2018) and **ALFRED** (Shridhar et al., 2020) on two benchmarks, D_{od} and D_e. We construct D_{od} for CEIF in open dynamic environments. It contains 120 instructions (60 per environment), covering four types of environmental changes and seven types of instruction complexity. We also use the ExRAP benchmark (**?**), denoted by D_e, for comparison with published results. On $\mathcal{D}_{od}$, we use GPT-5.4-mini as the backbone and compare RoboState with SOTA **SayCan**, **LLM-Planner**, **FLARE**, and **ExRAP**. On $\mathcal{D}_e$, we follow the original ExRAP settings using LLaMA-3-8B, and compare RoboState with **ZSP**, **SayCan**, **ProgPrompt**, **LLMPlanner**, and **ExRAP**. We evaluate overall effectiveness using success rate (**SR**) and pending steps (**PS**). For component-level analysis, we use goal-interpretation accuracy (**GoalAlign**), identity-alignment accuracy (**SlotID**), -hallucination rate (textbfHalluc.), and clarification cost (**ClarCost**). We average results over six seeds, with more than 6,480 runs per method. Appendix **??** provides the details for all datasets, baselines and metrics.

4.1 Overall Effectiveness

We evaluate RoboState on D_{od} along three dimensions.

Dynamic difficulty. We define dynamic difficulty by the frequency of environmental changes. Under low, medium, and high difficulty, instruction-triggering conditions and human-induced environmental events(e.g., object movement, occlusion, removal) occur at intervals of 8, 6, and 4 timesteps. Higher difficulty therefore requires the agent to align its environmental knowledge with true environment more robustly. As shown in columns 7-9 of Table 1, RoboState achieves the highest SR and the lowest PS across different difficulty. Specifically, RoboState improves SR by 47.22% and reduces PS by 4.70 timesteps on average, compared to the strongest baseline SayCan. FLARE's SR drops to 0.00%, as it lacks environmental tracking. These results show RoboState's robustness to frequent human-induced changes in dynamic environment.

Instruction scale. columns 1-3 of Table 1 also reports performance under medium dynamic difficulty as the number of parallel continual instructions increases from 3 to 7. As the instruction scale increases, the agent must monitor more triggering conditions and coordinate more pending tasks. RoboState outperforms all baselines in terms of SR at each instruction scale. Specifically, RoboState improves SR by 31.18%, compared with the strongest baseline ExRAP. Particularly, RoboState's relative advantage grows with instruction scale.

Table 1: Performance (**SR**↑ / **PS**↓) of RoboState over D_{od} with respect to dynamic difficulty, instruction scale and type.

Method	Scale (#inst.)			Instruction Type			Dynamic Difficulty		
	3	5	7	Sent.	Summ.	Vague	L	M	H
SayCan	26.67% / 5.00	16.00% / 4.25	8.57% / 8.00	21.67% / 2.08	20.00% / 3.67	16.67% / 2.10	41.67% / 8.52	35.00% / 14.29	33.33% / 18.30
LLMPlanner	33.33% / 8.00	16.00% / 7.75	11.43% / 5.50	50.00% / 4.30	41.67% / 3.64	31.67% / 4.00	38.33% / 7.00	20.00% / 11.50	18.33% / 16.00
FLARE	26.67% / 5.50	16.00% / 6.50	11.43% / 7.50	31.67% / 5.05	30.00% / 4.94	28.33% / 4.94	16.67% / 6.90	0.00% / –	0.00% / –
ExRAP	53.33% / 8.00	52.00% / 8.69	31.43% / 9.27	56.67% / 3.74	55.00% / 4.00	26.67% / 3.75	33.33% / 5.45	31.67% / 13.95	30.00% / 13.06
RoboState	80.00% / 8.17	76.00% / 9.53	74.29% / 9.85	86.67% / 5.02	78.33% / 4.98	75.00% / 7.69	86.67% / 5.02	83.33% / 9.02	81.67% / 12.98

(Average standard deviation is 5.67% (SR) / 3.63 (PS) over 6 seeds)

Table 2: Performance (**SR**↑ / **PS**↓) of RoboState over D_e with respect to non-stationarity, instruction scale and instruction type.

Method	Non-stationarity			Scale (#inst.)			Instruction Type		
	L	M	H	3	5	7	Sent.	Summ.	Vague
ZSP	20.59% / 31.03	20.06% / 32.06	17.28% / 24.08	33.11% / 22.86	20.06% / 32.06	7.43% / 59.16	–	–	–
SayCan	35.12% / 21.67	33.69% / 21.81	27.33% / 16.18	40.58% / 19.76	33.69% / 21.81	23.04% / 37.05	33.69% / 21.81	32.66% / 22.13	23.89% / 28.03
ProgPrompt	32.10% / 18.84	30.51% / 23.43	27.19% / 18.60	43.15% / 19.99	30.51% / 23.43	21.20% / 29.00	–	–	–
LLMPlanner	40.97% / 17.61	39.89% / 15.93	34.60% / 14.94	49.28% / 17.13	39.89% / 15.93	31.82% / 17.63	39.89% / 15.93	37.19% / 16.45	30.88% / 19.09
ExRAP	61.12% / 11.75	55.14% / 11.33	50.12% / 8.61	67.77% / 12.29	55.14% / 11.33	53.86% / 8.76	55.14% / 11.33	53.11% / 10.42	50.26% / 13.51
RoboState	88.60% / 8.08	87.72% / 9.01	80.70% / 9.50	85.09% / 8.54	81.58% / 11.83	79.82% / 7.79	88.60% / 8.08	83.33% / 9.39	75.44% / 5.49

(Average standard deviation is 5.52% (SR) / 3.45 (PS) over 6 seeds)

The SR improvement increases from 26.67% at scale 3 to 42.86% at scale 7. While RoboState requires slightly more PS than baselines, higher SR indicates that these PS are critical and necessary for CEIF success. These results indicate that RoboState effectively coordinates multiple instructions during their joint execution.

Instruction type. We further evaluate RoboState across three instruction types under medium dynamic difficulty. Sentence-wise type expresses each instruction in a distinct sentence. Summarized type uses more compact descriptions, whereas vague type refers to target objects abstractly. RoboState achieves the highest SR for all three types. Specifically, RoboState improves SR by 33.89% on average, compared with the strongest baseline ExRAP. While baselines seem to use less PSs, they sacrifice the SR effectiveness severely. The results present that RoboState is effective in inferring the instruction intention and grounds vague references to concrete objects.

Cross-benchmark evaluation on ExRAP. We evaluate RoboState on the ExRAP dataset under its benchmark settings. The evaluation follows ExRAP's original dimensions of non-stationarity, instruction scale, and instruction type. As presented in Table 2, RoboState outperforms ExRAP across all settings. Compared to ExRAP, RoboState improves SR by 30.21%, 23.24%, and 29.62% and reduces PS by 1.70, 1.41, and 4.10 timesteps for non-stationarity, instruction scale, and instruction type, respectively.

4.2 Ablation study

For each module, we construct ablated versions of RoboState, i.e., one module-level ablated version that removes the entire module and multiple component-level ablated versions that each remove one core component while retaining the rest of the module.

Impact of hierarchical reasoning for goal states. We establish the *w/o Goal Reasoning* ablated version by removing this entire module and passing the instruction directly to the planner. Further, we develop the *w/o Intention reasoning* ablated version through removing intention-reasoning component and utilizing LLM to infer the goal predicates directly from the instruction. We establish the *w/o Parameter acquisition* version by retaining intention reasoning but removing parameter acquisition. As shown in rows 3-6 of Table 3, removing Goal Reasoning decreases SR by 58.34%, and lowers GoalAlign from 87.50% to 56.52%. Further, the *w/o Intention reasoning* and *w/o Parameter acquisition* versions reduce SR by 60.00% and 55.00%, and reduce GoalAlign by 25.00% and 17.93%.

Impact of State-consistent Memory. We establish the *w/o State-consistent Memory* ablated version by removing the memory module and utilizing the LLM to answer environment queries from the current observation. Further, we develop the *w/o Memory Structure* ablated version by representing the memory as unstructured text instead of graph-structured relations. Moreover, we construct the *w/o State Alignment* version by retaining

Table 3: Ablation results for modules and components integrated by RoboState over D_{od}.

Module	Version	SR (%)↑	PS↓	GoalAlign (%)↑
Goal Reasoning	**RoboState (Full)**	86.67%	5.02	87.50%
	w/o Goal Reasoning	28.33%	7.41	56.52%
	w/o Intention Reasoning	26.67%	6.62	62.50%
	w/o Parameter Acquisition	31.67%	6.58	69.57%
				SlotID (%)↑
State Consistent Memory	**RoboState (Full)**	86.67%	5.02	100.00%
	w/o Memory	11.67%	5.77	18.18%
	w/o Memory Structure	21.67%	5.29	69.23%
	w/o State Alignment	43.33%	5.50	64.29%
				Halluc. (%)↓
STG Planning	**RoboState (Full)**	86.67%	5.02	0.00%
	w/o STG Planning	33.33%	6.93	14.15%
	w/o STG Construction	51.67%	6.61	7.09%
	w/o Path Merging	78.33%	6.72	0.00%

Average standard deviation: 4.86% (SR) / 1.56 (PS) over 6 seeds.

Table 4: The results across RoboState's versions with different LLMs, for evaluating the applicability to different LLMs.

Model	SR (%)↑	PS↓
GPT-5.4-mini	86.67%	5.02
Claude Haiku 4.5	83.33%	5.62
Gemini 3.5 Flash	84.62%	4.92
Llama 3.1 8B Instruct	81.58%	7.56
Qwen3 8B	79.23%	6.46

Average standard deviation: 5.02% (SR) / 1.36 (PS) over 6 seeds.

Table 5: Clarification results on Class G-(2) of D_{od}.

Metric	$B=0$	$B=1$	$B=2$	$B=3$
SR (%)↑	22.22%	66.67%	88.89%	88.89%
ClarCost	0.00	0.89	1.33	1.44

the graph structure but updating observations directly to memory without state-trajectory alignment. As shown in rows 8-11 of Table 3, *w/o State-consistent Memory* version reduces SR by 75.00% and increases PS by 0.75 timesteps, while decreases SlotID from 100.00% to 18.18%. Moreover, removing Memory Structure or State Alignment decreases SR by 65.00% and 43.34%, increases PS by 0.27 and 0.48 timesteps, and decreases SlotID by 30.77% and 35.71%, respectively.

Impact of STG-based Task Planning We establish the *w/o STG Planning* ablated version through removing the entire module and utilizing the LLM to generate the complete skill sequence. And we develop the *w/o STG construction* ablated version by omitting graph construction, using the LLM to generate the skill sequence for subtasks and reducing redundancy via path merging. The *w/o Path Merging* ablated version retains STG construction and path search but directly concatenates the paths without merging. As shown in rows 13-16 of Table 3, removing STG Planning decreases SR by 53.34%, increases PS by 1.91 timesteps, and raises Halluc. to 14.15%. Removing STG Construction decreases SR by 35.00 %, increases PS by 1.59 timesteps, and raises Halluc. to 7.09%. The *w/o Path Merging* decreases SR by 8.34% and increases PS by 1.70 timesteps.

4.3 Robustness and Cost

Backbone Robustness. We evaluate RoboState with multiple LLMs under the default setting, covering open and closed source models of different families and scales. As shown in Table 4, RoboState remains effective regardless of model types and scales. This robustness is consistent with the design of RoboState, which uses the LLM for semantic interpretation while constraining module outputs through explicit structures. This reduces reliance on unconstrained LLM generation and limits sensitivity to LLM choice.

Clarification Cost. We conduct experiment on the parameter-missing subset designed to require clarification. We vary the clarification budget b, the maximum number of questions permitted during each execution. A simulated user answers each query using the annotated ground-truth parameters. As b increases, SR largely saturates at $b = 2$. These results show that clarification is an effective extension. Comparison experiments in Section 4.1 use $b = 0$, their results is not attributed to additional user information.

5 Conclusion

In this paper, we propose RoboState, a state-aware planning framework for CEIF in open dynamic environments. We evaluate RoboState on both a new benchmark comprising 120 CEIF instructions and the existing ExRAP benchmark. The results demonstrate its robustness across diverse CEIF scenarios and its advantages over representative state-of-the-art task-planning approaches.

References

M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

M. Chang, G. Chhablani, A. Clegg, M. Dallaire Cote, R. Desai, M. Hlavac, V. Karashchuk, J. Krantz, R. Mottaghi, P. Parashar, et al. Partnr: A benchmark for planning and reasoning in embodied multi-agent tasks. In *International Conference on Learning Representations*, volume 2025, pages 65205–65268, 2025.

S. Chen, A. Xiao, and D. Hsu. Llm-state: Open world state representation for long-horizon task planning with large language model. *arXiv preprint arXiv:2311.17406*, 2023.

X. Chen, Y. Chen, Y. Fu, N. Gao, J. Jia, W. Jin, H. Li, Y. Mu, J. Pang, Y. Qiao, et al. Internvla-m1: A spatially guided vision-language-action framework for generalist robot policy. *arXiv preprint arXiv:2510.13778*, 2025.

W. Huang, P. Abbeel, D. Pathak, and I. Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022a.

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*, 2022b.

B. Kim, J. Kim, Y. Kim, C. Min, and J. Choi. Context-aware planning and environment-aware memory for instruction following embodied agents. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10936–10946, 2023.

T. Kim, B. Kim, and J. Choi. Multi-modal grounded planning and efficient replanning for learning embodied agents with a few examples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 4329–4337, 2025.

A. Kurenkov, M. Lingelbach, T. Agarwal, E. Jin, C. Li, R. Zhang, L. Fei-Fei, J. Wu, S. Savarese, and R. Martın-Martın. Modeling dynamic environments with scene graph memory. In *International Conference on Machine Learning*, pages 17976–17993. PMLR, 2023.

J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pages 9493–9500. IEEE, 2023.

K. Lyu, K. Wu, P. Li, W. Song, D. Wu, J. He, J. Chen, N. Yang, Z. Han, K. Luo, et al. From instruction following to cognitive navigation: A survey on the evolution of vision-and-language navigation. 2026.

T. Ohno. *Toyota production system: beyond large-scale production*. Productivity press, 2019.

A. O. Onishchenko, A. K. Kovalev, and A. I. Panov. Lookplangraph: Embodied instruction following method with vlm graph augmentation. *arXiv preprint arXiv:2512.21243*, 2025.

X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8494–8502, 2018.

K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. *arXiv preprint arXiv:2307.06135*, 2023.

G. Ratnayaka. Learning partial graph matching via optimal partial transport. *arXiv preprint arXiv:2410.16718*, 2024.

A. Z. Ren, A. Dixit, A. Bodrova, S. Singh, S. Tu, N. Brown, P. Xu, L. Takayama, F. Xia, J. Varley, et al. Robots that ask for help: Uncertainty alignment for large language model planners. *arXiv preprint arXiv:2307.01928*, 2023.

G. Sarch, Y. Wu, M. Tarr, and K. Fragkiadaki. Open-ended instructable embodied agents with memory-augmented large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3468–3500, 2023.

M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2998–3009, 2023.

F. Wang, S. Lyu, P. Zhou, A. Duan, G. Guo, and D. Navarro-Alarcon. Instruction-augmented long-horizon planning: Embedding grounding mechanisms in embodied mobile manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 14690–14698, 2025.

Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, and Y. Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*, 2023.

Z. Wu, Z. Wang, X. Xu, H. Yin, Y. Liang, A. Ma, J. Lu, and H. Yan. Embodied instruction following in unknown environments. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 21825–21832. IEEE, 2025.

P. Xu, J. Zheng, and Y. Mu. Roboagent: Chaining basic capabilities for embodied task planning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15276–15290, 2026.

Y. Yang, T. Zhou, K. Li, D. Tao, L. Li, L. Shen, X. He, J. Jiang, and Y. Shi. Embodied multi-modal agent trained by an llm from a parallel textworld. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 26275–26285, 2024.

M. Yoo, J. Jang, W.-J. Park, and H. Woo. Exploratory retrieval-augmented planning for continual embodied instruction following. *Advances in Neural Information Processing Systems*, 37:67034–67060, 2024.

A Appendix

A.1 Metrics.

We use two metrics to evaluate overall effectiveness. *Success rate* (SR) is the proportion of completed tasks among CEIF instances. *Pending steps* (PS) is the number of timesteps required by the generated plan. To evaluate the impact of individual components, we introduce four fine-grained metrics. *GoalAlign* measures goal-interpretation accuracy as the proportion of predicted goal states that match the annotations. *SlotID* measures identity-alignment accuracy as the proportion of post-perturbation observations assigned to the correct object slot. *Halluc.* measures the skill-hallucination rate as the proportion of robot-unsupported skills in generated plans. *ClarCost* measures the clarification cost as the average number of questions per execution.

A.2 Datasets.

We collect the following datasets for evaluation.

Open dynamic CEIF Dataset (D_{od}). To evaluate CEIF of open dynamic environments systematically, we construct the first benchmark for this setting. We construct 60 CEIF instructions in each of VirtualHome and ALFRED, with 120 instructions in total. The scenarios cover two complementary dimensions of open-dynamic CEIF, including open-dynamic environmental changes and instruction-processing complexity. The environmental dimension comprises four categories, (1) *existence*, where objects temporarily disappear and later reappear or are relocated, (2) *identity*, where visually similar instances must be distinguished, (3) *state*, where object states change between observation intervals, (4) *rule*, where environmental rules are temporarily introduced and later removed. The instruction dimension covers three stages of CEIF processing. For trigger evaluation, the scenarios cover instructions with (5) normally satisfiable triggering conditions and (6) ambiguous referents requiring user clarification. For goal-feasibility reasoning, they include instructions with (7) infeasible conditions due to missing object capabilities and (8) unavailable key elements requiring an alternative strategy. For execution planning, they cover (9) resource-contention instructions requiring action serialisation or object substitution, (10) parallel multi-object cases permitting action merging, and (11) multistep tasks requiring objects to undergo prerequisite state transitions. According to these scenarios, we simulate open dynamics (human intervention) by utilizing scripts to modify the simulator state between consecutive agent observations.

ExRAP (D_e). ExRAP provides only CEIF benchmark containing 19 VirtualHome instructions and 16 ALFRED instructions. Evaluation on ExRAP enables direct comparison with published results under the original benchmark setting.

A.3 Baselines.

We first compare RoboState with four representative baselines over D_{od} in Table 1. SayCan and LLMPlanner are the classic baselines for embodied instruction following. Both are included in the ExRAP evaluation, providing common reference points across Table 1 and Table 2. Since ExRAP directly addresses CEIF, we use it as our primary baseline. We reproduced it after consulting the authors about implementation details. FLARE is the SOTA study for addressing plan-state discrepancies, whose setting closely matches our open dynamic setting. Together, these methods cover general planning, CEIF, and dynamic planning.

Moreover, we compare RoboState with ZSP, SayCan, ProgPrompt, LLMPlanner, and ExRAP on D_e in Table 2. All methods are run under the original ExRAP settings.

Our evaluation is broader and more systematic than SOTA CEIF studies. First, D_{od} is the largest dataset with 120 instructions, whereas ExRAP has 35. Second, extensive evaluation (more than 6,480 runs for each tool) spans multiple difficulty, scales, types and seeds for systematic conclusions.