BEYOND SHARED PLANS: ALIGNMENT-CONSTRAINED TREE SEARCH FOR EMBODIED COLLABORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) based embodied agents increasingly use shared plans to coordinate long-horizon collaboration. However, a coherent plan does not guarantee coherent execution: agents may progress at different rates, causing ongoing work to be reassigned, completed work to remain active, or new actions to conflict with the current physical state. We study this planning–execution mismatch as the *cognition–execution gap* and propose *Alignment-Constrained Tree Search (ACTS)*. ACTS connects collaborative planning with ongoing embodied execution through three steps: (1) *Planning with Persistent Responsibilities* keeps each agent's active assignment, grounded target, and execution stage across planning calls; (2) *Search with Responsibility Constraints* admits only continuations supported by current execution evidence before reflection and expansion, while using unmet conditions to revise rejected proposals; and (3) *Updating Responsibilities after Execution* uses observed progress and confirmed effects to close completed work, preserve compatible ongoing work, and update the state for subsequent planning. Together, ACTS turns repeated plan revision into a closed planning–execution loop in which collaborative search remains consistent with physical progress. Experiments on TDW-MAT and C-WAH validate ACTS's planning–execution alignment and demonstrate competitive task performance on both benchmarks.

1 INTRODUCTION

Large language models (LLMs) give embodied agents useful abilities for reasoning, communication, and high-level planning (Huang et al., 2022; Ichter et al., 2023; Song et al., 2023). In multi-agent tasks, however, a shared plan unfolds through many physical actions. Agents therefore reach different execution stages: one may finish an assignment while another is still carrying an object or approaching a receptacle. A later planning call must account for this unfinished work rather than treating every revision as a fresh allocation.

Recent cooperation methods communicate local observations, negotiate assignments, and revise joint plans as the task changes (Zhang et al., 2024; Mandi et al., 2024; Liu et al., 2025; Zu et al., 2025). CoTS additionally searches several planner–critic refinements and decides when to update a selected plan (Zu et al., 2025). These methods improve plan formation, but plan revision alone does not identify the execution status of each assignment. Replacing work that is already progressing wastes useful motion; retaining work after its required effect has occurred leaves the next plan stale. What is missing is a persistent link between the shared plan and the physical work that each agent is carrying out.

Figure 1 shows the distinction. Bob is already transporting objects while Alice becomes available for a new target. Bob's assignment should continue if current evidence still supports it, while Alice may receive new work. When a required physical effect is later confirmed, only the associated assignment should close. We call inconsistencies along the plan–responsibility–action–effect chain the *cognition–execution gap*. Section 3 defines four interfaces at which this mismatch can appear: preconditions, action realization, observed effects, and the update of the next planning state.

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.

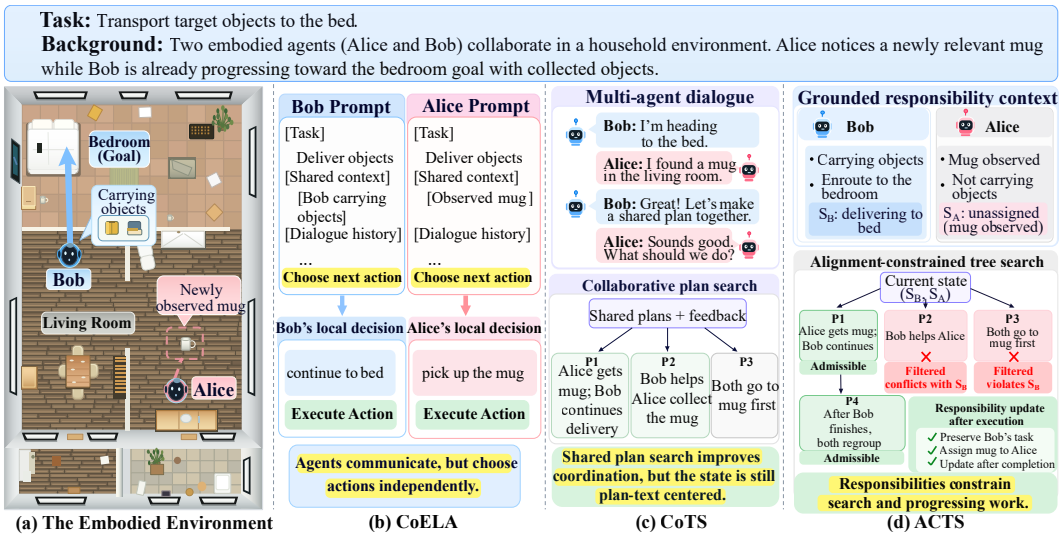


Figure 1: **From shared plans to continuing work.** CoELA chooses local actions and CoTS searches shared plans. ACTS additionally keeps the currently authorized responsibilities in the search state, so a new assignment can be added without discarding compatible work already in progress. The displayed scenario and scores are schematic.

We propose *Alignment-Constrained Tree Search (ACTS)*. At each planning boundary, ACTS records the current responsibilities and the evidence relevant to them. A candidate enters reflection and expansion only if its immediate assignments are supported and jointly compatible. Rejected candidates leave explicit unmet requirements that can guide a revised proposal. After executing a selected continuation, ACTS uses observed progress, confirmed terminal effects, and authorized replacement to update each responsibility. The updated state becomes the root context for the next search.

Our contributions are:

- **A planning–execution formulation.** We represent active work as persistent responsibilities and define four interfaces at which a collaborative plan can lose support from execution evidence.
- **Search and update mechanisms.** ACTS constrains candidate admission before reflection, uses unmet conditions to revise unsupported proposals, and updates responsibilities from execution outcomes without overwriting compatible work that is still progressing.
- **Task and mechanism evaluation.** We evaluate ACTS on TDW-MAT and C-WAH and compare it with CoTS and a matched PCE baseline. Trace audits, fixed-context information tests, an admission-placement control, an unmet-condition revision ablation, and recorded responsibility updates examine how the method behaves beyond final task scores.

2 Related Work

LLM-based embodied cooperation. Recent embodied-cooperation work uses local observations and teammate information to coordinate under partial observation (Puig et al., 2021; Zhang et al., 2024; Chen et al., 2025; Kang et al., 2025; Chang et al., 2025; Zhang et al., 2025a). General LLM multi-agent frameworks also study role-based collaboration and group coordination (Li et al., 2023; Chen et al., 2024). Language-based embodied systems add communication and joint planning: RoCo and CaPo coordinate through plan discussion and updates, REVECA combines adaptive planning with trajectory validation, and CoTS searches planner–critic refinements (Mandi et al., 2024; Liu et al., 2025; Seo et al., 2025; Zu et al., 2025). PCE represents uncertainty in planning assumptions and uses it to guide embodied planning (Seo et al., 2026). ACTS addresses a different part of the loop: after a plan changes, it records which assignments are still active, which may be replaced, and which have enough physical evidence to close.

Grounding, feedback, and verification. Language plans become useful only when tied to executable choices. Multimodal embodied models connect language with perception and action (Driess et al., 2023; Zitkovich et al., 2023; Jiang et al., 2023), while Code as Policies and ProgPrompt map language to executable programs or situated action structures (Liang et al., 2023; Singh et al., 2023). Other work uses affordances, symbolic scene descriptions, spatial value maps, or constrained decoding to ground decisions (Ichter et al., 2023; Rana et al., 2023; Huang et al., 2023a;b). Feedback-based agents revise decisions after interaction and retain information across attempts (Huang et al., 2023c; Yao et al., 2023b; Kim et al., 2025; Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024), while tree methods explore alternative continuations (Yao et al., 2023a; Zhou et al., 2024). Symbolic and verification-oriented planners expose preconditions, effects, and subtask checks (Garrett et al., 2020; Liu et al., 2023; Zhang et al., 2025b; Xu et al., 2025). ACTS organizes these ingredients around responsibility persistence across planning calls: current evidence restricts which collaborative branches may be searched, and later execution evidence updates the same responsibility records.

3 Problem Formulation

Task state and evidence. We consider n cooperating embodied agents under partial observation in a decentralized setting (Bernstein et al., 2002; Zhang et al., 2024); the experiments use two agents. Agent i has local history $h_t^i = (o_{0:t}^i, \mu_{0:t}^i, a_{0:t-1}^i)$, where μ_t^i contains received messages. A joint action $\mathbf{a}_t$ induces $s_{t+1} \sim T(\cdot \mid s_t, \mathbf{a}_t)$. For counted goal units $\mathcal{G} = \{g_1, \dots, g_K\}$, evaluator progress is

$$\mathbf{q}_t = \big(\mathbb{I}[s_t \models g_1], \dots, \mathbb{I}[s_t \models g_K]\big). \tag{1}$$

The planner does not observe s_t directly. It uses permitted evidence E_t, with source and observation time retained, and maintains $\bar{\mathbf{q}}_t$ only from goals supported by that evidence. Planned effects therefore remain distinct from observed possession or placement.

Responsibilities. A shared plan P_t proposes a division of work. ACTS represents its physically active part by $\boldsymbol{\kappa}_t = (\kappa_t^1, \dots, \kappa_t^n)$. Each responsibility binds an agent to an action family, grounded target, destination or goal unit, and execution stage. Its lifecycle records whether the work is progressing, closed, superseded, or failed. Primitive commands may change while the responsibility remains the same; changing its target or goal creates a new assignment that requires a fresh check. The maintained state is

$$Z_t = (P_t, \boldsymbol{\kappa}_t, \bar{\mathbf{q}}_t). \tag{2}$$

A responsibility is deliberately coarser than a primitive command but more concrete than free-form plan text. For example, one delivery responsibility may persist across navigation, approach, and release while its agent, grounded target, destination, and goal relation remain unchanged. Changing any of those bindings creates new work that must be checked again. This granularity lets ACTS preserve useful physical progress without treating every low-level action as a new assignment.

Appendix A gives the complete responsibility record and evidence rules.

Candidate continuations. A candidate b specifies which responsibilities to retain, replace, or start. A collaborative tree $\mathcal{T}_t = (\mathcal{V}_t, \mathcal{E}_t)$ contains these alternatives. A selected branch is converted into grounded work and then executable actions:

$$b_t^\star \to P_{b_t^\star} \to \boldsymbol{\kappa}_{b_t^\star} \to \mathbf{a}_t \to E_{t+1}. \tag{3}$$

Plan parsing maps the selected responsibilities to executable commands while preserving the authorized agent, target, and intended effect. The next observation then updates the responsibility whose contract was active rather than treating the full plan text as a new physical state.

Four cognition–execution interfaces. Let Pre, Real, Prog, and Close denote current-stage support, compatible commands, permitted intermediate progress, and terminal closure. For agent i, let $r_{t,\tau}^i$ be the relation due at interface $\tau \in \{\mathrm{pre}, \mathrm{real}, \mathrm{eff}, \mathrm{upd}\}$: respectively $E_t \models \mathrm{Pre}(\kappa_t^i)$, $a_t^i \in \mathrm{Real}(\kappa_t^i, E_t)$, $\mathrm{Prog}(\kappa_t^i, E_t, E_{t+1}) = 1 \vee E_{t+1} \models \mathrm{Close}(\kappa_t^i)$, and $\mathrm{Cons}_i(Z_{t+1}, E_{t+1}) = 1$. For each interface, let $E_{t,\tau}$ denote the permitted evidence available when that relation is evaluated.

Its evidence status and gap indicator are

$$E_{t,\tau} = \begin{cases} E_t, & \tau \in \{\text{pre}, \text{real}\}, \\ \langle E_t, E_{t+1} \rangle, & \tau = \text{eff}, \\ E_{t+1}, & \tau = \text{upd}, \end{cases}$$
$$s^i_{t,\tau} = \text{Supp}\big(r^i_{t,\tau}; E_{t,\tau}\big), \tag{4}$$
$$d^i_{t,\tau} = \begin{cases} 0, & s^i_{t,\tau} = \text{supported}, \\ 1, & s^i_{t,\tau} = \text{contradicted}, \\ \text{NA}, & s^i_{t,\tau} = \text{unknown}. \end{cases}$$

Each relation is evaluated only at its proper boundary: admission or activation, command dispatch, due effect verification, or state update. Only a contradiction produces an executed-interface gap; unknown evidence may block a new assignment when support is required, but it is not itself recorded as physical failure. Thus, an unfinished transport is not a failure while its contract admits the observed progress. New inconsistencies enter the persistent set

$$\mathcal{D}_{t+1} = \text{Active}\big(\mathcal{D}_t \cup \Delta^{\text{new}}_{t+1}; Z_{t+1}, E_{t+1}\big), \quad \Delta^{\text{new}}_{t+1} = \{(i, \tau, \kappa^i_t) : d^i_{t,\tau} = 1\}. \tag{5}$$

A record leaves $\mathcal{D}$ when the violated relation is restored or the responsibility is validly retired. Candidate rejections are stored separately in $\mathcal{J}$ because a proposal that was never executed does not constitute an executed-effect gap.

4 Alignment-Constrained Tree Search

4.1 Planning with Persistent Responsibilities

At a planning boundary, ACTS first reconciles existing responsibilities with the latest evidence. The planner then proposes joint continuations; admission decides which proposals may be reflected and expanded, and failed conditions can be used to construct a revised proposal. The selected continuation is checked again before activation. During execution, primitive commands remain attached to their responsibility. The returned evidence advances, closes, supersedes, or preserves that work before the next search. The resulting loop is search $\rightarrow$ execute $\rightarrow$ update responsibilities $\rightarrow$ search, rather than a sequence of unrelated plan rewrites.

Figure 2 shows where the method enters the CoELA/CoTS pipeline (Zhang et al., 2024; Zu et al., 2025). Perception provides task-relevant observations and action outcomes; memory keeps the plan, task record, responsibilities, and unresolved requirements; ACTS uses them during collaborative search; execution binds selected work to grounded targets and returns evidence for the next update. Appendix B gives the records passed across these interfaces.

4.2 Search with Responsibility Constraints

Let the root context be $C_t = \langle Z_t, E_t, \mathcal{D}_t \rangle$, with rechecked proposal requirements $\mathcal{J}_t$ supplied separately. A candidate b proposes immediate responsibilities $\boldsymbol{\kappa}_b$ and may also describe later conditional work. Using the compact notation below (expanded in Appendix C), admission depends only on requirements that are due now:

$$\text{Adm}(b; C_t, \mathcal{J}_t) = \mathbb{I}[\text{current preconditions, ownership, and joint compatibility are supported}]. \tag{6}$$

Future steps do not borrow their own predicted effects as evidence. For example, if Bob is already delivering x and Alice can see but does not hold y, a proposal that makes Alice deliver y immediately is rejected. A revision may instead assign Alice to acquire y while Bob continues x. The root evidence remains fixed throughout this search call.

ACTS stores a failed proposal in $\mathcal{J}$ with its grounded identity and unmet requirements. ACTS rechecks these requirements after relevant evidence changes, and they can guide a new proposal. ACTS therefore uses rejected proposals as structured search feedback: unsupported branches are excluded before ordinary reflection, while their unmet requirements can guide subsequent proposal generation. The searchable set is

$$\mathcal{V}^{\text{adm}}_t = \{v \in \mathcal{V}_t : \text{Adm}(v; C_t, \mathcal{J}_t) = 1\}. \tag{7}$$

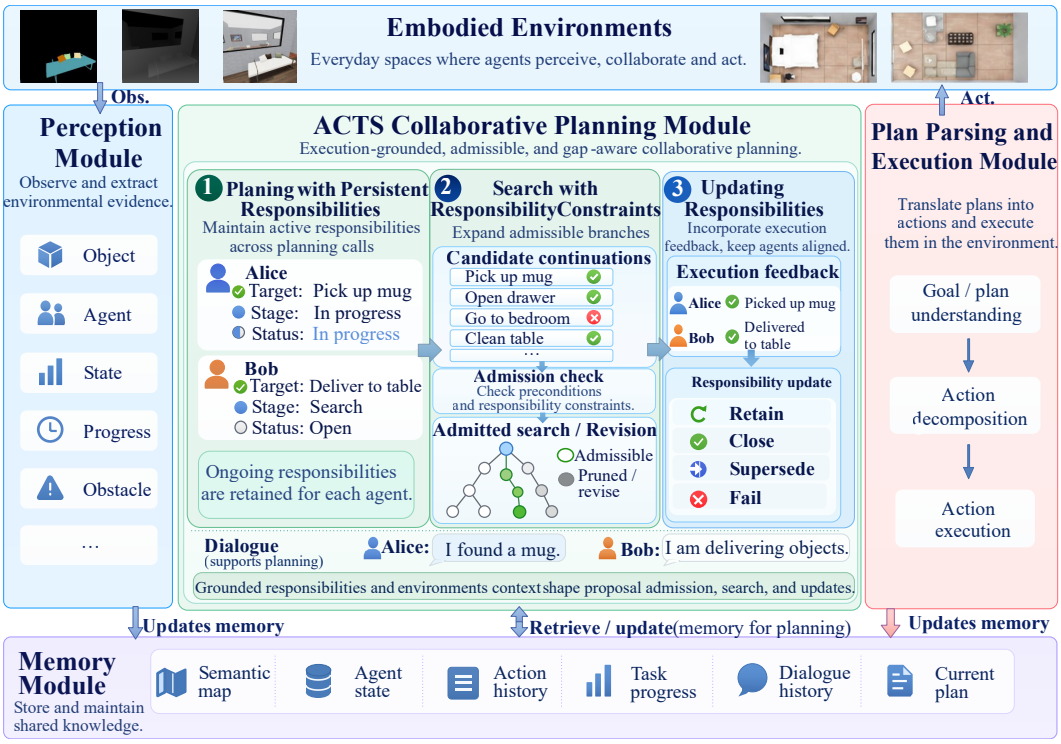


Figure 2: **ACTS framework.** Responsibilities are maintained inside collaborative search. Current evidence restricts candidate admission; admitted branches are searched; execution feedback then updates the corresponding responsibilities.

Within this set, ACTS retains the planner–critic value and visit-based selection inherited from CoTS. Admission and value serve different roles: admission tests whether the current execution state supports a continuation, whereas critic value ranks only the continuations that pass this test—for a fixed evaluated candidate pool with deterministic eligibility, filtering before or after ranking yields the same constrained maximizer. The distinction matters online because reflection and expansion consume a finite budget, so early admission changes which branches receive evaluation. Search is finite, and previously evaluated branches may be reused only if the evidence on which their admission depended remains valid. Appendix C gives the exact admission factors, UCT rule, backup, reuse conditions, and bounded recovery.

4.3 Updating Responsibilities after Execution

Before activation, the selected immediate responsibility is checked against fresh evidence. A compatible progressing responsibility can continue; a changed target, goal relation, or destination requires a new admission check. After the environment step, ACTS associates the execution receipt with the same responsibility and evaluates its contract $\Gamma(\kappa) = \langle \mathrm{Pre}, \mathrm{Real}, \mathrm{Prog}, \mathrm{Close} \rangle$. Intermediate progress keeps the responsibility active. Confirmed closure marks only that the work is complete. Authorized replacement is recorded as supersession rather than physical completion.

The joint update can be written as

$$(Z_{t+1}, \mathcal{D}_{t+1}, \mathcal{J}_{t+1}) = F(Z_t, \mathcal{D}_t, \mathcal{J}_t, \mathbf{a}_t, E_{t+1}), \tag{8}$$

where F first incorporates the execution evidence, reconciles affected responsibilities and task progress, and then rechecks retained search requirements. This order is important when one agent finishes while another remains in progress. The next planning call receives the updated state, so closure of one responsibility does not silently erase compatible work owned by its teammate. Algorithms 1–2 in Appendix C give the full search and update procedures; Figure 6 provides the detailed search illustration.

5 EXPERIMENTS

We test three levels of evidence: task performance, whether the execution context supports recorded proposals, and whether the proposed search/update mechanisms behave as intended. The mechanism analyses use distinct units (proposal traces, fixed candidate contexts, matched executions, and responsibility events), with each unit interpreted at its corresponding analysis level.

5.1 EXPERIMENTAL SETUP

We follow the TDW-MAT and C-WAH protocols used by CoELA and CoTS (Zhang et al., 2024; Zu et al., 2025). TDW-MAT contains 24 Food/Stuff tasks with a 3,000-frame horizon and is evaluated by Transport Rate (TR) under oracle and non-oracle perception. C-WAH contains ten tasks from five activity types with a 250-step horizon; we report mean completion steps for visual and symbolic observations. Appendix D gives the observation/action interfaces and aggregation rules.

Comparisons include RHP/CoRHP, MHP/CoMHP, MAT, CoELA, RoCo, CaPo, PCE, CoTS, and CoTS-S. CoTS-S keeps the CoTS planner while using the executor shared with ACTS. Published and rerun values are separated in Appendix D.2. Locally evaluated benchmark methods use three independent runs with matched task seeds across methods, and Tables 1 and 2 report the mean performance across runs. Experiments use GPT-4 or GPT-3.5-Turbo as indicated (OpenAI, 2023), temperature 0.7, and a 512-token output limit. Mechanism analyses use candidate setting 3 and tree height 2. Control definitions and runtime settings are in Appendices D.4 and D.5.

Table 1: TDW-MAT transport rate (TR, %, ↑). O/NO: oracle/non-oracle perception. * published by CoELA; ‡ RoCo values reported by CoTS; † our evaluation, reported as the mean over three runs with matched task seeds. Subscripts show relative change from CoELA. Best and second-best distinct LLM results per row are bold and blue.

	Base Agent		RL	LLM Agent: GPT-3.5						
Setting	RHP*	CoRHP*	MAT*	CoELA*	RoCo‡	CaPo†	PCE†	CoTS†	CoTS-S†	ACTS†
O: Stuff	49	74	17	73	$80_{+9.59\%}$	$82_{+12.33\%}$	–	$81_{+10.96\%}$	$80_{+9.59\%}$	$\mathbf{84}_{+15.07\%}$
O: Food	52	76	13	72	$74_{+2.78\%}$	$\mathbf{86}_{+19.44\%}$	–	$83_{+15.28\%}$	$83_{+15.28\%}$	$84_{+16.67\%}$
O: Avg.	50	75	15	73	$77_{+5.48\%}$	$\mathbf{84.0}_{+15.07\%}$	–	$82.0_{+12.33\%}$	$81.5_{+11.64\%}$	$\mathbf{84.0}_{+15.07\%}$
NO: Stuff	36	54	–	39	$47_{+20.51\%}$	$45_{+15.38\%}$	–	$\mathbf{50}_{+28.21\%}$	$48_{+23.08\%}$	$49_{+25.64\%}$
NO: Food	49	67	–	67	$64_{-4.48\%}$	$\mathbf{70}_{+4.48\%}$	–	$65_{-2.99\%}$	$65_{-2.99\%}$	$68_{+1.49\%}$
NO: Avg.	43	61	–	52	$55_{+5.77\%}$	$57.5_{+10.58\%}$	–	$57.5_{+10.58\%}$	$56.5_{+8.65\%}$	$\mathbf{58.5}_{+12.50\%}$
	Base Agent		RL	LLM Agent: GPT-4						
Setting	RHP*	CoRHP*	MAT*	CoELA*	RoCo‡	CaPo†	PCE†	CoTS†	CoTS-S†	ACTS†
O: Stuff	49	74	17	83	$85_{+2.41\%}$	$86_{+3.61\%}$	$86_{+3.61\%}$	$91_{+9.64\%}$	$90_{+8.43\%}$	$\mathbf{94}_{+13.25\%}$
O: Food	52	76	13	87	$89_{+2.30\%}$	$90_{+3.45\%}$	$89_{+2.30\%}$	$92_{+5.75\%}$	$89_{+2.30\%}$	$\mathbf{96}_{+10.34\%}$
O: Avg.	50	75	15	85	$87_{+2.35\%}$	$88.0_{+3.53\%}$	$87.5_{+2.94\%}$	$91.5_{+7.65\%}$	$89.5_{+5.29\%}$	$\mathbf{95.0}_{+11.76\%}$
NO: Stuff	36	54	–	61	$64_{+4.92\%}$	$64_{+4.92\%}$	$73_{+19.67\%}$	$77_{+26.23\%}$	$76_{+24.59\%}$	$\mathbf{85}_{+39.34\%}$
NO: Food	49	67	–	82	$80_{-2.44\%}$	$83_{+1.22\%}$	$83_{+1.22\%}$	$\mathbf{88}_{+7.32\%}$	$86_{+4.88\%}$	$86_{+4.88\%}$
NO: Avg.	43	61	–	71	$72_{+1.41\%}$	$73.5_{+3.52\%}$	$78.0_{+9.86\%}$	$82.5_{+16.20\%}$	$81.0_{+14.08\%}$	$\mathbf{85.5}_{+20.42\%}$

5.2 BENCHMARK TASK PERFORMANCE

Table 1 reports TDW-MAT transport. With GPT-4, ACTS averages 95.0% TR with oracle perception and 85.5% without it, compared with 91.5%/82.5% for CoTS and 87.5%/78.0% for PCE. The improvement is greatest on Stuff, while non-oracle Food remains slightly below CoTS (86% vs. 88%), showing that the overall gain is driven more strongly by Stuff. With GPT-3.5, ACTS reaches 84.0%/58.5%. The gains under both perception settings suggest that ACTS remains useful even when object identities are directly available, because possession, ongoing work, and delivery effects must still be maintained across planning calls. On C-WAH, ACTS averages 77 visual and 50 symbolic steps (Table 2), improving over CoTS at 80/55. Relative to PCE, ACTS is better in the visual setting (77 vs. 88). In contrast, PCE is better in the symbolic setting (45 vs. 50).

This contrast indicates that relative performance depends on the observation setting, where symbolic observations reduce perceptual grounding burden. Across both benchmarks, ACTS uses the same search and responsibility-update logic with different execution contracts. TDW-MAT verifies delivery at the goal, whereas C-WAH verifies requested ON/IN relations after execution. Thus, the adapter changes evidence and closure predicates while preserving the planning rule, allowing the same alignment mechanism to operate under distinct task semantics.

Table 2: C-WAH task performance under GPT-4: average completion steps (↓). Locally evaluated † entries are means over three runs with matched task seeds; * and ‡ are published values as defined in Table 1. Subscripts denote relative reduction from CoELA. Best results are bold and second-best distinct results are blue.

	Base Agent		LLM Agent: GPT-4						
Setting	MHP*	CoMHP*	CoELA*	RoCo‡	CaPo†	PCE†	CoTS†	CoTS-S†	ACTS†
Visual	141	103	92	$89_{+3.26\%}$	$86_{+6.52\%}$	$88_{+4.35\%}$	$80_{+13.04\%}$	$85_{+7.61\%}$	$\mathbf{77_{+16.30\%}}$
Symbolic	111	75	57	$57_{+0.00\%}$	$55_{+3.51\%}$	$\mathbf{45_{+21.05\%}}$	$55_{+3.51\%}$	$53_{+7.02\%}$	$50_{+12.28\%}$

5.3 Ablation and Mechanism Analysis

Trace-level evidence of proposal–execution mismatch. We examine whether collaborative search generates proposals unsupported by the current execution state. The analysis includes all 145 deduplicated ordinary search–candidate pairs from the TDW-MAT raw post-search guard. Each proposal is evaluated against the root evidence available at generation time, considering only the immediate joint step. Recovery proposals, future observations, and final episode outcomes are excluded. We record search-time rejections in $\mathcal{J}$, whereas executed-interface inconsistencies in Section 3 are recorded in the executed-gap set $\mathcal{D}$.

Table 3: **Trace-level proposal–execution mismatch in collaborative search.** Counts are deduplicated ordinary search–candidate pairs evaluated using only the root evidence available at generation time and the immediate joint step. Categories are mutually exclusive. We exclude recovery proposals; we report the eligibility checker separately because it evaluates a broader set of constraints.

Audit category	Count	Fraction
All reflected ordinary proposals	145	100.0%
Lack sufficient support for the immediate action	**79**	**54.5%**
Unsupported task precondition or grounding	38	26.2%
Replacement of a progressing responsibility	16	11.0%
Unresolved cleanup or detachment	6	4.1%
Insufficient current execution evidence	19	13.1%
Immediate action supported	66	45.5%
Eligibility checker flagged invalid	143	98.6%

Among the 145 reflected proposals, 79 (54.5%) lack sufficient support for the immediate action: 38 involve unsupported task preconditions or grounding, 16 replace protected progressing work, six omit required cleanup, and 19 lack decisive execution evidence. The remaining 66 are supported at the immediate step. A fixed-seed manual review reproduces all 30 sampled labels. The broader eligibility checker flags 143/145 because it additionally evaluates later steps and implementation constraints, making this a broader eligibility statistic than the immediate-step mismatch rate. Appendix D.7 details the labeling and sensitivity rules.

Ablation of execution-context evidence. To isolate which execution information supports continuation decisions, we keep 24 candidate contexts fixed and vary only the evidence shown to a blinded offline assessor. Coverage is the fraction receiving a definite *admissible/inadmissible* judgment; no independent gold label is available for these contexts. Plan/task text alone yields 4.2% coverage. Providing the full execution record raises coverage to 95.8% in text form and 87.5% in the structured form. Within the structured representation, removing ownership, progress history, or

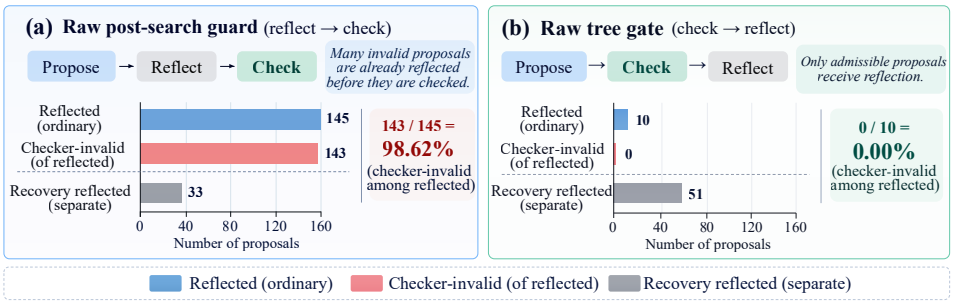


Figure 3: **Effect of admission placement on reflection.** Counts are deduplicated ordinary search–candidate pairs. Checker-invalid candidates form a subset of reflected candidates, while recovery proposals are reported separately and excluded from both ratios. The raw tree gate isolates the effect of applying the same eligibility check before reflection and expansion.

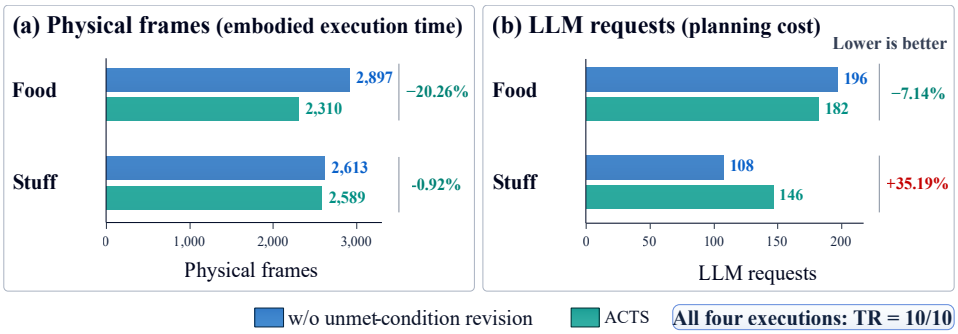


Figure 4: **Ablation of unmet-condition revision.** Both variants complete 10/10 objects (TR=100%). ACTS reduces completion time in both paired executions, while language-model requests vary across Food and Stuff.

closure evidence reduces coverage to 70.8%, 83.3%, and 79.2%, respectively. Ownership produces the largest drop, while progress and closure evidence also affect whether a continuation can be resolved from the current execution state. Appendix D.8 gives the sampling and projection rules, and Appendix E.3 reports all six conditions.

Effect of admission placement. The raw post-search guard and raw tree gate apply the same eligibility criterion on opposite sides of ordinary reflection. The raw post-search guard reflects 145 ordinary candidates, of which 143 are checker-invalid, whereas the raw tree gate reflects ten, none of which are checker-invalid (Figure 3). Moving admission before reflection therefore prevents unsupported ordinary proposals from consuming planner–critic evaluation and redirects the finite search budget toward eligible continuations. The placement control also separates early filtering from the remaining ACTS mechanisms: in the representative Stuff execution, the raw tree gate completes 5/10 objects, whereas ACTS completes 10/10. ACTS additionally revises proposals from unmet conditions and reconciles responsibilities after execution. Appendix E.2 reports the corresponding execution outcomes.

Ablation of unmet-condition revision. Figure 4 compares ACTS with the control that disables proposal revision from unmet conditions. Each Food/Stuff pair shares the same task and seed, and all four executions deliver 10/10 objects. On Food, ACTS reduces completion time from 2,897 to 2,310 frames while decreasing model requests from 196 to 182. On Stuff, completion time changes from 2,613 to 2,589 frames, while requests increase from 108 to 146. The paired results show that unmet-condition revision can shorten physical execution while preserving full task completion, although its planning cost remains task-dependent.

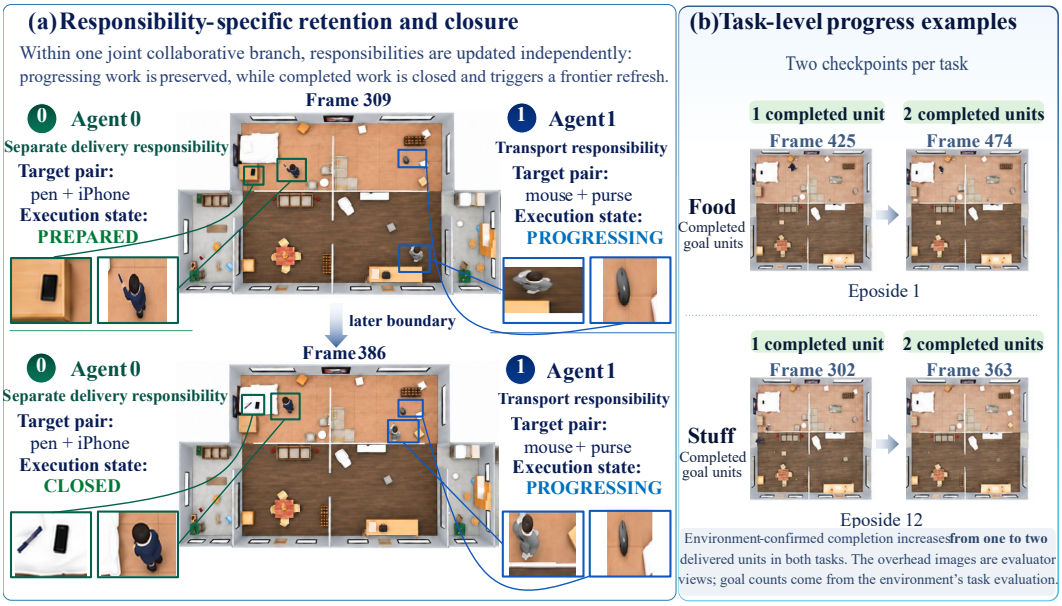


Figure 5: **Responsibility maintenance and task progress.** (a) Agent 0's delivery closes after confirmation at frame 318 and is reconciled at frame 386, while Agent 1's progressing transport is retained. (b) Separate Food and Stuff executions advance from one to two environment-confirmed goal units. Views are external evaluator images.

5.4 Responsibility Maintenance During Execution

Responsibility-specific retention and closure. Across the complete Food and Stuff ACTS pairs, 62/63 searches admit a new continuation, and eight environment-confirmed completions are matched to the responsibility that closes. Figure 5(a) illustrates the responsibility-specific update: Agent 0's delivery closes after environment confirmation, while Agent 1's independent transport remains PROGRESSING. Panel (b) shows task-level progression in separate Food and Stuff executions, where evaluator-confirmed goal units increase from one to two. Together, these records show that ACTS can close completed work without resetting a compatible teammate responsibility.

Mechanism evidence across planning and execution. Taken together, the analyses connect the observed proposal mismatch to ACTS behavior across successive stages. The execution-context ablation identifies which state information makes continuation decisions determinate, while the placement control shows how early admission redirects planner–critic evaluation toward eligible branches. The revision ablation further shows how unmet conditions reshape subsequent planning and physical completion time. Finally, the responsibility traces show that confirmed execution effects close the corresponding work while preserving compatible teammate progress. These results directly connect search-time constraints, proposal revision, and execution-time state maintenance within the ACTS loop.

6 Conclusion

This paper introduces ACTS, an alignment-constrained tree search framework that connects collaborative replanning with persistent responsibilities during embodied execution. By constraining candidate continuations with evidence, revising unsupported proposals from unmet requirements, and updating responsibilities after execution, ACTS maintains consistency between shared plans and physical progress. Experiments on TDW-MAT and C-WAH demonstrate competitive performance against planning baselines, while trace analyses characterize the roles of execution context, early admission, revision, and responsibility updates. **Limitation.** ACTS is evaluated with two agents and benchmark-specific execution contracts; extending it to larger teams, models, and broader environments remains future work.

AI USE STATEMENT

Generative AI tools were used solely to assist with language editing and polishing, including improving grammar, clarity, and readability. All research ideas, methodology, experiments, analyses, results, and conclusions were developed and verified by the authors.

REPRODUCIBILITY STATEMENT

Sections 3–4 define the gap interfaces, responsibility state, branch admission, and execution update. Appendices A–C give the evidence assumptions, contracts, search/update procedures, and planning interfaces. Appendix D records benchmark protocols, baseline provenance, controls, and runtime settings. Appendices D.7 and D.8 specify the trace audit and fixed-context analysis. Appendix E provides the execution records used by the mechanism analyses. Published reference results remain separated from our evaluations.

REFERENCES

Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operations Research*, 27(4): 819–840, 2002.

Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, Siddharth Patki, Ishita Prasad, Xavier Puig, Akshara Rai, Ram Ramrakhya, Daniel Tran, Joanne Truong, John M. Turner, Eric Undersander, and Tsung-Yen Yang. PARTNR: A benchmark for planning and reasoning in embodied multi-agent tasks. In *International Conference on Learning Representations*, 2025.

Junting Chen, Checheng Yu, Xunzhe Zhou, Tianqi Xu, Yao Mu, Mengkang Hu, Wenqi Shao, Yikai Wang, Guohao Li, and Lin Shao. EMOS: Embodiment-aware heterogeneous multi-robot operating system with LLM agents. In *International Conference on Learning Representations*, 2025.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, 2024.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8469–8488. PMLR, 2023.

Chuang Gan, Siyuan Zhou, Jeremy Schwartz, Seth Alter, Abhishek Bhandwaldar, Dan Gutfreund, Daniel L. K. Yamins, James J. DiCarlo, Josh McDermott, Antonio Torralba, and Joshua B. Tenenbaum. The ThreeDWorld transport challenge: A visually guided task-and-motion planning benchmark towards physically realistic embodied AI. In *IEEE International Conference on Robotics and Automation*, pp. 8847–8854, 2022. doi: 10.1109/ICRA46639.2022.9812329.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pp. 440–448, 2020. doi: 10.1609/icaps.v30i1.6739.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9118–9147. PMLR, 2022.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3D value maps for robotic manipulation with language models. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 540–562. PMLR, 2023a.

Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, and Brian Ichter. Grounded decoding: Guiding text generation with grounded models for embodied agents. In *Advances in Neural Information Processing Systems*, volume 36, 2023b.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 1769–1782. PMLR, 2023c.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T. Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as I can, not as I say: Grounding language in robotic affordances. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 2023.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. VIMA: Robot manipulation with multimodal prompts. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 14975–15022. PMLR, 2023.

Li Kang, Xiufeng Song, Heng Zhou, Yiran Qin, Jie Yang, Xiaohong Liu, Philip H. S. Torr, Lei Bai, and Zhenfei Yin. VIKI-R: Coordinating embodied multi-agent cooperation via reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3935.

Jinyeon Kim, Cheolhong Min, Byeonghwi Kim, and Jonghyun Choi. Pre-emptive action revision by environmental feedback for embodied instruction following agents. In *Proceedings of the 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2396–2428. PMLR, 2025.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, pp. 51991–52008, 2023.

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. In *2023 IEEE International Conference on Robotics and Automation*, pp. 9493–9500, 2023. doi: 10.1109/ICRA48891.2023.10160591.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

Jie Liu, Pan Zhou, Yingjun Du, Ah-Hwee Tan, Cees G. M. Snoek, Jan-Jakob Sonke, and Efstratios Gavves. CaPo: Cooperative plan optimization for efficient embodied multi-agent cooperation. In *International Conference on Learning Representations*, 2025.

Zhao Mandi, Shreeya Jain, and Shuran Song. RoCo: Dialectic multi-robot collaboration with large language models. In *IEEE International Conference on Robotics and Automation*, pp. 286–299, 2024. doi: 10.1109/ICRA57147.2024.10610855.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. VirtualHome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8494–8502, 2018.

Xavier Puig, Tianmin Shu, Shuang Li, Zilin Wang, Yuan-Hong Liao, Joshua B. Tenenbaum, Sanja Fidler, and Antonio Torralba. Watch-and-help: A challenge for social perception and human-AI collaboration. In *International Conference on Learning Representations*, 2021.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. SayPlan: Grounding large language models using 3D scene graphs for scalable robot task planning. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 23–72. PMLR, 2023.

SeungWon Seo, SeongRae Noh, Junhyeok Lee, SooBin Lim, Won Hee Lee, and HyeongYeop Kang. REVECA: Adaptive planning and trajectory-based validation in cooperative language agents using information relevance and relative proximity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 23295–23303, 2025. doi: 10.1609/aaai.v39i22.34496.

SeungWon Seo, SooBin Lim, SeongRae Noh, Haneul Kim, and HyeongYeop Kang. From assumptions to actions: Turning LLM reasoning into uncertainty-aware planning for embodied agents. In *International Conference on Learning Representations*, 2026.

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

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation*, pp. 11523–11530, 2023. doi: 10.1109/ICRA48891.2023.10161317.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu 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*, pp. 2998–3009, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024.

Muning Wen, Jakub Kuba, Runji Lin, Weinan Zhang, Ying Wen, Jun Wang, and Yaodong Yang. Multi-agent reinforcement learning is a sequence modeling problem. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16509–16521, 2022.

Tianyang Xu, Dan Zhang, Kushan Mitra, and Estevam Hruschka. Verification-aware planning for multi-agent systems. *arXiv preprint arXiv:2510.17109*, 2025.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b.

Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. In *International Conference on Learning Representations*, 2024.

Hongxin Zhang, Zeyuan Wang, Qiushi Lyu, Zheyuan Zhang, Sunli Chen, Tianmin Shu, Behzad Dariush, Kwonjoon Lee, Yilun Du, and Chuang Gan. COMBO: Compositional world models for embodied multi-agent cooperation. In *International Conference on Learning Representations*, 2025a.

Xiaopan Zhang, Hao Qin, Fuquan Wang, Yue Dong, and Jiachen Li. LaMMA-P: Generalizable multi-agent long-horizon task allocation and planning with LM-driven PDDL planner. In *IEEE International Conference on Robotics and Automation*, 2025b.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160. PMLR, 2024.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of the 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023.

Lizheng Zu, Lin Lin, Song Fu, Na Zhao, and Pan Zhou. Collaborative tree search for enhancing embodied multi-agent collaboration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 29513–29522, 2025.

A TASK AND RESPONSIBILITY SEMANTICS

This appendix expands the objects used in Section 3. It distinguishes the environment's task progress, information available to the agents, and the responsibilities authorized for execution.

A.1 INFORMATION ACCESS AND TASK PROGRESS

Each agent acts from local observations and received messages. A planning call may combine facts shared through the permitted coordination interface, but evaluator records are not planning inputs. Observations, reports, and action outcomes keep their source and time. TDW-MAT counts requested units delivered before its frame horizon (Gan et al., 2022; Zhang et al., 2024), whereas C-WAH measures steps to satisfy the requested relations (Puig et al., 2021; Zhang et al., 2024). The evaluator state $\mathbf{q}_t$ and planner record $\bar{\mathbf{q}}_t$ are therefore distinct: a message or intention can update the planning context without proving a physical goal.

Table 4: Information used by planning and evaluation.

Quantity	Meaning for planning	Role in evaluation
Local observation	Current sensor/symbolic facts	What was available at the decision
Teammate report	Received fact or intention with source/time	Information age and interpretation
Action outcome	Command status and subsequent observation	Link action to physical effect
Planner progress $\bar{\mathbf{q}}_t$	Goals supported by permitted evidence	What later decisions may assume
True progress $\mathbf{q}_t$	Evaluator-defined goal satisfaction	Benchmark performance

A.2 RESPONSIBILITY IDENTITY AND EXECUTION STAGE

A responsibility is the part of a shared plan that is currently assigned to an agent. We write

$$\kappa = \langle i, \omega, g, \mathbf{x}, d, \sigma, \Gamma_\kappa \rangle, \tag{9}$$

where i is the agent, ω the action family, g the goal unit, $\mathbf{x}$ the grounded target instance or bundle, d the destination/receptacle, σ the execution stage, and Γ_κ the verification contract. Responsibility identity follows the assigned work, not its wording. Navigation, approach, and release may therefore constitute a single delivery responsibility. Changing the target, destination, or goal relation creates new work that requires admission. Acquisition and delivery remain separate: possession can close acquisition without satisfying the delivery goal.

Planning boundary t is also distinct from the environment clock f_t. Several planner–critic refinements may occur at one boundary, so tree depth records proposal refinement rather than completed physical actions.

A.3 THE FOUR INTERFACES AND EVIDENCE STATUS

The indicators in Equation (4) are evaluated only where their relation is due. **Preconditions** ask whether current evidence supports the stage to be activated. **Realization** asks whether the issued command is compatible with the same agent, target, goal, and stage. **Effects** distinguish permitted intermediate progress from terminal closure. **Updates** check that relevant responsibility and task facts remain consistent after the outcome is incorporated. Thus a release command does not by itself prove delivery, and an inactive candidate has no executed-effect test.

For a required relation ϕ, let $\mathrm{Supp}(\phi; E) \in \{\mathsf{supported}, \mathsf{contradicted}, \mathsf{unknown}\}$. Current work is admitted only when its required facts are supported. A contradiction can justify revising the affected responsibility; unknown evidence can require inspection or deferral. Missing observations do not automatically establish physical failure.

A.4 Reconciliation and Search Rejections

An execution-gap record in $\mathcal{D}$ stores its interface, agent, grounded responsibility, evidence, and first applicable boundary. It is reconciled when the relation is restored, or the responsibility is validly retired. Supersession does not rewrite a missing physical effect as completed.

Search rejections $\mathcal{J}$ have a different role. A candidate enters $\mathcal{J}$ when its immediate assignment fails admission. Because that proposal was not executed, the rejection alone does not create an execution event in $\mathcal{D}$. After an environment step, ACTS updates Z, reconciles $\mathcal{D}$, and rechecks the retained requirements in $\mathcal{J}$. Rewording a candidate does not remove an unmet grounded requirement. Appendix D.9 describes how to reconstruct these interfaces from traces.

A.5 Admission Property

At a fixed root, every admitted child satisfies the implemented admission relation by construction; activation then rechecks the selected work against fresh evidence. For a fixed candidate pool with fixed values and deterministic eligibility, the filter position does not change the constrained maximum. Formally, let $\mathcal{B}$ be finite and assume at least one candidate satisfies Adm. Filtering first selects an element of

$$\arg \max_{b \in \mathcal{B}: \mathrm{Adm}(b)=1} V(b). \tag{10}$$

If we instead sort all candidates by V and scan until the first eligible one, every preceding element is ineligible and no eligible candidate has a larger value; the same constrained maximum is attained (up to the same tie rule). This observation is deliberately limited to a fixed pool. In online tree search, reflection and expansion consume a finite budget, so early admission changes which candidates are generated and evaluated. Figure 3 measures that search-placement effect.

B Module Interfaces and Execution Contracts

ACTS retains the perception–memory–planning–execution organization of CoELA and CoTS (Zhang et al., 2024; Zu et al., 2025), but adds records that persist across planning calls. Perception supplies observed entities, relations, holding state, and action outcomes with their source and time. Memory keeps the current plan, task record, active responsibilities, execution gaps $\mathcal{D}_t$, and proposal requirements $\mathcal{J}_t$. Planning proposes and evaluates continuations only after current assignments pass admission. Execution binds a selected assignment to a grounded target, checks it again against fresh evidence, and returns progress or closure evidence.

B.1 Evidence, Memory, Planning, and Execution Roles

Evidence. ACTS does not replace the benchmark perception stack. It consumes the facts that stack makes available and keeps the boundary at which each fact was observed. In TDW-MAT this includes visible/recognized objects, pose, holding state, messages, and action outcomes; in C-WAH it uses the corresponding symbolic or visual interface. Reports from a teammate remain reports rather than being silently converted into physical effects.

Memory. The persistent addition is the responsibility record. It links a selected branch to the agent, target, goal relation, stage, and expected evidence that later execution must satisfy. The plan and dialogue history remain available, but they do not replace this binding. Execution-gap records $\mathcal{D}_t$ and rejected proposal requirements $\mathcal{J}_t$ are stored separately because only the former describe inconsistencies that have reached an execution boundary.

Planning. The planner proposes a complete joint continuation. Immediate assignments are checked against the root evidence and current ownership before they receive reflection or expansion. Later steps stay conditional. If a proposal fails, ACTS can reuse the unmet requirement when asking for another proposal, but the revised branch must pass the same admission test. Planning value is therefore used to compare supported alternatives, not to override an unsupported physical condition.

Execution and update. The selected branch is parsed into work for each agent and checked again before activation. Navigation or approach can preserve the same responsibility across several primitive commands. The update then associates the action receipt and new evidence with that responsibility. Acquisition closes on confirmed possession; delivery or placement closes only on the specified task relation. Compatible work owned by the other agent is left unchanged.

The shared contract is

$$\Gamma(\kappa) = \langle \mathrm{Pre}_\kappa, \mathrm{Real}_\kappa, \mathrm{Prog}_\kappa, \mathrm{Close}_\kappa \rangle, \tag{11}$$

which separates what must hold before activation, what commands may realize the responsibility, what intermediate evidence counts as progress, and what observation closes it.

Table 5: Records carried across search and execution.

Record	Information retained	Decision role
Root evidence	Entities, relations, source, observation time	Grounding and freshness
Active responsibility	Agent, goal, target, stage	Continuation and ownership
Execution history	Confirmed progress and effects	Retention and closure
Candidate continuation	Immediate assignments and conditional successors	Admission and search
Unresolved requirement	Missing/contradicted relation and evidence	Revision or later recheck
Search statistics	Visits, value, tree relations	Selection and backup

Table 6: Responsibility contracts used in the experiments.

Responsibility	Support before activation	Progress evidence	Closure
Acquire	Bound target, capacity, compatible ownership	Approach/manipulation	Intended object held
Deliver	Goal-relevant possession and destination	Navigation/approach	Delivery relation confirmed
Place on/in	Intended object held, receptacle bound	Access/opening/approach	Requested placement relation
Inspect	Grounded observation target	Arrival or usable view	Required evidence obtained
Explore	Region and information need	Movement/scan coverage	Exploration criterion met

The contract families are shared; their predicates are implemented with each benchmark's native observations and actions. Adapting ACTS therefore requires mappings for targets, action families, progress, and closure, but not a different search or update rule. Section D records the two benchmark mappings used here.

C Search Procedure, Constraints, and Prompt Interfaces

This appendix details the planning module in Section 4.2. ACTS retains the proposal–feedback–valuation structure of CoTS (Zu et al., 2025). It uses the execution context to govern candidate admission, selection, and reuse. The templates below specify the planning interfaces and their inputs from execution evidence.

C.1 Root Evidence and the Meaning of a Search Edge

At the start of each search call, memory supplies $C_t = \langle Z_t, E_t, \mathcal{D}_t \rangle$ and rechecked rejection memory $\mathcal{J}_t$. A node stores $(P_v, \kappa_v, \mu_v, Q_v, N_v)$: its joint plan, proposed responsibilities, planner–critic dialogue, running value, and visits. The root context and input $\mathcal{J}_t$ remain fixed throughout the call. A working set $\mathcal{J}$ starts from $\mathcal{J}_t$ and accumulates unmet requirements from candidate checks. It supplies feedback for proposal generation. Each edge refines a proposal under this fixed evidence. New possession and delivery facts require execution evidence.

For "acquire y, then deliver y," acquisition is the immediate responsibility. Delivery remains conditional on a later observed grasp. Search checks the immediate assignment against current evidence and retains the future prerequisite for later activation. This separation allows the sequence to be planned while preserving its execution order. Section C.8 gives a detailed example. The selected plan undergoes fresh activation checks before execution.

C.2 Constraint Construction and Admission at the Current Stage

A candidate $b = (P_b, \kappa_b^{\text{now}}, \mathcal{S}_b^{\text{cond}})$ separates immediate responsibilities from conditional successors. Its semantic identity records each agent, action family, grounded target, destination, and goal relation. This identity stays the same under paraphrasing.

Current requirements and retained requirements. $\mathcal{K}^{\text{task}}(E_t, Z_t)$ contains requirements implied by the current task and evidence. These cover entity binding, capacity, stage-specific possession, destination support, remaining goals, and joint allocation. A gap record $d \in \mathcal{D}_t$ stores the affected responsibility, executed interface, and unresolved relation ρ_d. A search rejection record $r \in \mathcal{J}_t$ stores the candidate's semantic identity and unmet admission requirements. It also records the evidence and responsibility fields each requirement depends on. The compilers Ξ and Ψ preserve each requirement only for candidates to which it applies.

Let $\text{Applies}(r, b)$ denote semantic relevance to the candidate's immediate assignment. Let $\chi_r(b; C_t)$ require that the assignment respects the retained relation. The history-dependent condition reads the two record sets separately:

$$H(b; C_t, \mathcal{J}_t) = \bigwedge_{d \in \mathcal{D}_t : \text{Applies}(d,b)} \chi_d(b; C_t) \wedge \bigwedge_{r \in \mathcal{J}_t : \text{Applies}(r,b)} \chi_r(b; C_t). \tag{12}$$

For example, a possession requirement still applies when immediate delivery is reworded. A candidate can instead propose supported acquisition and keep delivery conditional. Execution releases the requirement once it establishes possession. Search rejections record unmet admission requirements. Executed-interface gaps record inconsistencies assessed at execution boundaries. Compilation and reconciliation preserve these distinct sources.

Admission components. Write G, S, R, and C_{joint} for grounded identity and effect specification, current-stage support, responsibility continuity, and joint compatibility, respectively. The implemented interface is expressed as

$$\text{Adm}(b; C_t, \mathcal{K}_t, \mathcal{J}_t) = G(b; E_t) \wedge S(b; E_t) \wedge R(b; Z_t, E_t) \wedge C_{\text{joint}}(b; Z_t, E_t) \wedge H(b; C_t, \mathcal{J}_t). \tag{13}$$

G requires bound entities and an effect that can verify each immediate assignment. S checks only those assignments. It retains the prerequisites of $\mathcal{S}_b^{\text{cond}}$ for later activation. R preserves the identity of compatible ongoing work. Replacement requires both an evidence-supported retirement or supersession decision and a separately supported new assignment. A higher planning score alone cannot authorize replacement. Confirmed closure, verified failure, or another evidence-supported invalidation can permit revision. Unobserved progress alone cannot justify abandonment. C_{joint} excludes competing ownership, incompatible agent allocations, and duplicated completed goals.

The check returns both admission and unmet requirements,

$$(\alpha_b, U_b) = \text{Check}(b, C_t, \mathcal{J}_t), \quad \alpha_b = \text{Adm}(b; C_t, \mathcal{K}_t, \mathcal{J}_t), \tag{14}$$

where U_b records whether each unmet support test is contradicted or unknown. Only admitted candidates receive an ordinary critic score. An unmet-condition revision uses U_b to change the immediate assignment. The revised candidate is checked under the same root evidence.

Table 7: Illustrative admission decisions for distinct grounded objects x and y. These examples specify semantics, not measured outcomes.

Evidence	Proposed continuation	Decision
Alice does not hold x	Deliver x immediately	Defer direct delivery.
x available; Alice has capacity	Acquire x, then deliver it	Check acquisition; retain conditional delivery.
Alice holds x and progresses	Alice continues; Bob acquires y	Retain Alice; check Bob.
Alice holds x	Bob acquires the same x	Exclude conflicting ownership.
Delivery of x confirmed	Treat x as undelivered	Reconcile the completed responsibility.
Location of x unknown	Grasp at an assumed location	Consider a grounded inspection.
Relevant condition remains contradicted	Repeat assignment under new wording	Retain the evidence-dependent rejection.

Release of an unresolved requirement. Reconciliation can be written as

$$\mathrm{Active}(\mathcal{D}; Z, E) = \{d \in \mathcal{D} : \neg\,\mathrm{Restored}(d; Z, E) \wedge \neg\,\mathrm{Retired}(d; Z, E)\}. \tag{15}$$

Restoration requires evidence supporting the affected relation. Retirement requires confirmed closure or authorized supersession of that responsibility. Rewording, unrelated observations, and tree reconstruction leave the requirement active. Search rejection records are stored separately and reconciled at a later execution boundary. Root evidence remains fixed throughout the search.

C.3 Planning Value, Selection, and Backup

ACTS retains the CoTS planner–critic valuation of work allocation and movement cost (Zu et al., 2025). This value compares admitted plans, with no added weighted gap score. Admission, current-plan assessment, and node valuation remain separate interfaces. Diagram scores are illustrative. The ordinary critic evaluates only admitted candidates.

For an eligible visited child v of p, exploration uses

$$\mathrm{UCT}(v) = Q_v + c\sqrt{\frac{\log N_{\mathrm{pa}(v)}}{N_v}}, \qquad N_v > 0. \tag{16}$$

Unvisited eligible children are considered before this expression is applied. Given a critic value r, each node on the selected ancestor path is updated by

$$N_v^+ = N_v + 1, \qquad Q_v^+ = \frac{N_v Q_v + r}{N_v + 1}. \tag{17}$$

Invalid candidates are excluded from the tree. An admitted candidate needs a usable critic evaluation to enter value-based final extraction. Failed or missing responses remain unavailable evaluations rather than measured zeros.

For evaluated candidates, final extraction has the constrained form

$$b^\star \in \arg\max_{b \in \mathcal{A}_t : \mathrm{Adm}(b; C_t, \mathcal{K}_t, \mathcal{J}_t) = 1} Q(v_b). \tag{18}$$

Here $\mathcal{A}_t$ is the evaluated set produced within the budget. $Q(v_b)$ is the backed-up value of the node generated by candidate b. The exploration bonus guides tree traversal; final extraction uses the planning value. Appendix D.5 reports the implementation settings. The fixed-pool proposition in Appendix A.5 identifies online search allocation as the relevant effect of admission placement. With the evaluated pool fixed, the constrained optimum is unchanged.

C.4 Reusing Search after Evidence Changes

Reuse separates a plan's semantic content from its search statistics. A cached plan must pass admission under the new C_t and rechecked $\mathcal{J}_t$ before selection or execution. Rejection lookup uses the candidate's semantic identity and each requirement's evidence and responsibility dependencies.

$\mathrm{RecheckRejections}(\mathcal{J}, C^+)$ reassesses every retained requirement against the reconciled state and evidence in C^+. Each decision depends on this current support. A requirement is removed when the evidence satisfies it or establishes valid retirement of the affected responsibility. Otherwise, it retains its current support record. A record stays active while at least one relevant requirement remains unmet. Thus, new possession evidence can release a delivery requirement while an unrelated ownership requirement stays active.

Algorithm 2 returns $C^+ = \langle Z^+, E^+, \mathcal{D}^+ \rangle$ and $\mathcal{J}^+$. These become C_{t+1} and $\mathcal{J}_{t+1}$ for the next search. Proposal generation and Check then use the retained requirements through Ψ and H. Unchanged dependencies preserve a rejection. Changed dependencies prompt reassessment, and remove a requirement only when the release conditions hold. Every reused candidate passes all current admission checks, including after an old requirement is released.

Previous candidate descriptions can serve as proposal records. The new execution context determines their eligibility. Search statistics belong to the current tree and cannot establish admission. The execution state and fresh evidence determine which responsibilities continue.

Budget and depth bounds apply to ordinary generation and unmet-condition revision. Each revision is checked as a new candidate under the fixed context. If no new candidate is admitted, compatible ongoing work can continue. An idle agent can receive a supported inspection or exploration through the same admission rule. If neither new nor retained work is supported, the procedure returns the unmet condition.

C.5 Search and Execution Procedure

Algorithm 1 specifies the search interface. Its inputs are a grounded context, candidate proposals, and bounded search parameters. These parameters set proposal width, depth limit, revision limit, exploration coefficient, and search budget. The maintained execution state initializes the root. Every expanded node stores a candidate plan and its responsibility state tied to execution evidence. Admission precedes tree entry, reflection, and value backup. UCT selects among eligible visited nodes. Final extraction selects from evaluated admissible candidates. Algorithm 2 handles physical execution updates separately.

A pending item (v, b, j) stores its parent, candidate, and the number of revisions already used. Parsing separates immediate responsibilities from conditional successors before admission. The working set starts at $\mathcal{J} \leftarrow \mathcal{J}_t$ and accumulates the grounded unmet conditions U_b produced by failed checks. $\mathrm{Propose}$ can use these conditions when generating a later alternative, but $\mathrm{Check}(b, C_t, \mathcal{J}_t)$ always evaluates the candidate against the fixed execution context of the current call. In particular, a proposed future grasp cannot update possession evidence until an environment step actually produces a new observation. The returned working set becomes $\mathcal{J}_t^{\mathrm{out}}$ for Algorithm 2. $\mathrm{MergeRejection}$ merges duplicate semantic keys while retaining the evidence and responsibility dependencies needed for later rechecking. Missing critic responses are retried within the configured bounds. Retain returns a jointly compatible set of supported existing responsibilities and leaves unsupported idle slots unassigned.

Algorithm 1 Alignment-constrained collaborative search at a fixed execution context

Require: Context $C_t = \langle Z_t, E_t, \mathcal{D}_t \rangle$; persistent rejection records $\mathcal{J}_t$;
candidate proposals $\mathcal{R}$; search parameters Ω
Ensure: Supported new/retained continuation $b^\star$ or $\perp$; accumulated rejection records $\mathcal{J} = \mathcal{J}_t^{\text{out}}$

$\mathcal{K}_t \leftarrow \mathcal{K}^{\text{task}}(E_t, Z_t) \cup \Xi(\mathcal{D}_t, E_t, Z_t) \cup \Psi(\mathcal{J}_t, E_t, Z_t)$
$\mathcal{T} \leftarrow \text{Root}(C_t; Q_0 = 0, N_0 = 0)$; $\mathcal{A} \leftarrow \varnothing$; $\mathcal{J} \leftarrow \mathcal{J}_t$
$\mathcal{Q} \leftarrow \{(\text{root}(\mathcal{T}), b, 0) : b \in \mathcal{R}\}$
while the search budget remains **do**
  if $\mathcal{Q} = \varnothing$ **then**
    $v \leftarrow \text{SelectExpandable}(\mathcal{T}, H, c, \prec)$
    if $v = \perp$ **then**
      break
    end if
    $\mathcal{B} \leftarrow \text{Propose}(P_v, \mu_v, C_t, \mathcal{J}, k)$
    $\mathcal{Q} \leftarrow \{(v, b, 0) : b \in \mathcal{B}\}$
    $\text{RecordExpansionAttempt}(v)$
    if $\mathcal{Q} = \varnothing$ **then**
      continue
    end if
  end if
  $(v, b, j) \leftarrow \text{Pop}_{\prec}(\mathcal{Q})$
  $(P_b, \kappa_b^{\text{now}}, \mathcal{S}_b^{\text{cond}}) \leftarrow \text{Parse}(b)$
  $(\alpha_b, U_b) \leftarrow \text{Check}(b, C_t, \mathcal{J}_t)$
  if $\alpha_b = 0$ **then**
    $\mathcal{J} \leftarrow \text{MergeRejection}(\mathcal{J}, \text{Key}(b), U_b, Z_t, E_t)$
    if $j < r_{\max}$ **then**
      $b' \leftarrow \text{Revise}(b, U_b, C_t)$
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(v, b', j+1)\}$
    end if
    continue
  end if
  $w \leftarrow \text{AddChild}(\mathcal{T}, v, b, C_t)$
  initialize $Q_w = N_w = 0$ for the admitted child w
  $(\mu_w, r_w) \leftarrow \text{Reflect}(b, C_t)$
  if $r_w \neq \perp$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{b_w\}$
    for all $u \in \text{Ancestors}(w) \cup \{w\}$ **do**
      $Q_u \leftarrow (N_u Q_u + r_w)/(N_u + 1)$; $N_u \leftarrow N_u + 1$
    end for
  end if
end while
if $\mathcal{A} \neq \varnothing$ **then**
  return $\text{TieBreak}_{\prec}(\arg\max_{b \in \mathcal{A}: \text{Adm}(b; C_t, \mathcal{K}_t, \mathcal{J}_t) = 1} Q(v_b)), \mathcal{J}$
end if
$b^{\text{keep}} \leftarrow \text{Retain}(C_t, \mathcal{K}_t, \mathcal{J}_t)$
return b^{keep} if supported, otherwise $\perp$; and $\mathcal{J}$

A revised candidate keeps its parent unless it is later admitted and expanded. Depth limits count admitted refinements, whereas physical execution advances only after search returns. Search descendants remain proposals under the same root evidence. Parsing, fresh activation, and dispatch preserve the grounded target binding; only Algorithm 2 can incorporate later execution receipts and physical evidence into the next context.

C.6 Verification, Closure, and Successor Activation

Before activation, the executor parses the selected plan's immediate responsibilities and checks them against fresh permitted evidence. Dispatch binds each issued command to its agent, target, stage, and intended effect. Changing the target or goal unit creates a new responsibility; a compatible command toward the same target can retain the existing one. If a new slot fails activation, the slot stays unassigned or preserves supported ongoing work. The resulting joint set is checked after this merge so that a newly activated assignment cannot invalidate a retained teammate responsibility.

At each responsibility-specific reconciliation boundary, Algorithm 2 receives the authorized state and the responsibilities actually bound at dispatch, κ^{exec}, rather than the last proposed text. The executor and reconciliation logic update stage σ when a primitive receipt or stage-specific evidence satisfies the contract for moving to the next stage; the revised σ is stored in κ^{+} and therefore in Z^{+}. Lifecycle status $\ell(\kappa)$ remains a separate quantity. The branch order in Algorithm 2 is intentional: terminal closure is handled first, authorized supersession second, accepted intermediate progress third, and a due verified-failure test fourth. This order specifies precedence without assuming that Prog and $\mathrm{VerifiedFailure}$ are logically mutually exclusive. Unknown evidence alone preserves the previous status. Only permitted evidence can update $\bar{\mathbf{q}}$; a confirmed grasp can close acquisition, but delivery still requires its own confirmed goal relation. Search rejections $\mathcal{J}$ remain separate from executed-interface gap records $\mathcal{D}$ because an unexecuted rejected proposal has no physical effect to verify.

Algorithm 2 Responsibility-specific reconciliation after execution evidence

Require: Authorized state $Z_t = (P_t, \boldsymbol{\kappa}_t, \bar{\mathbf{q}}_t)$; $E_t, \mathcal{D}_t$;
dispatch bindings κ^{exec}, commands $\mathbf{a}$, receipts $\mathbf{r}$, new evidence E^{+};
authorized supersessions $\mathcal{S}$ and search output $\mathcal{J} = \mathcal{J}_t^{\mathrm{out}}$
Ensure: Updated $Z^{+}, \mathcal{D}^{+}, C^{+}, \mathcal{J}^{+}$ and responsibilities requiring a fresh successor check
$\boldsymbol{\kappa}^{+} \leftarrow \boldsymbol{\kappa}_t$; $\mathcal{F} \leftarrow \varnothing$
for all responsibility records $\kappa^{i} \in \boldsymbol{\kappa}_t$ affected by E^{+}, $\mathbf{r}$, or $\mathcal{S}$ **do**
  $e_i \leftarrow \mathrm{BindEvidence}(E^{+}, \mathbf{r}, \mathrm{Key}(\kappa^{i}))$
  if $e_i \models \mathrm{Close}(\kappa^{i})$ **then**
    $\ell(\kappa^{i,+}) \leftarrow$ CLOSED; $\mathcal{F} \leftarrow \mathcal{F} \cup \{i\}$
  else if $\mathrm{AuthorizedSupersession}(\kappa^{i}, \mathcal{S}, e_i)$ **then**
    $\ell(\kappa^{i,+}) \leftarrow$ SUPERSEDED
  else if $\mathrm{Prog}(\kappa^{i}, E_t, e_i) = 1$ **then**
    $\sigma(\kappa^{i,+}) \leftarrow \mathrm{AdvanceStage}(\kappa^{i}, \mathbf{r}, e_i)$; preserve identity and progressing status
  else if $\mathrm{Due}(\kappa^{i}) \wedge \mathrm{VerifiedFailure}(\kappa^{i}, e_i)$ **then**
    $\ell(\kappa^{i,+}) \leftarrow$ FAILED
  else
    $\ell(\kappa^{i,+}) \leftarrow \ell(\kappa^{i})$
  end if
end for
$\bar{\mathbf{q}}^{+} \leftarrow \mathrm{IncorporateConfirmedGoals}(\bar{\mathbf{q}}_t, E^{+})$
$P^{+} \leftarrow \mathrm{ReconcilePlan}(P_t, \boldsymbol{\kappa}^{+}, \bar{\mathbf{q}}^{+})$; $Z^{+} \leftarrow (P^{+}, \boldsymbol{\kappa}^{+}, \bar{\mathbf{q}}^{+})$
$\mathcal{G}^{+} \leftarrow \mathrm{EvaluateDueInterfaces}(\kappa^{\mathrm{exec}}, \mathbf{a}, E_t, E^{+}, Z^{+})$
$\mathcal{D}^{+} \leftarrow \mathrm{Active}(\mathcal{D}_t \cup \mathcal{G}^{+}; Z^{+}, E^{+})$
$C^{+} \leftarrow \langle Z^{+}, E^{+}, \mathcal{D}^{+} \rangle$
$\mathcal{J}^{+} \leftarrow \mathrm{RecheckRejections}(\mathcal{J}, C^{+})$
Store $(C^{+}, \mathcal{J}^{+})$ as $(C_{t+1}, \mathcal{J}_{t+1})$ for the next search
return $Z^{+}, \mathcal{D}^{+}, C^{+}, \mathcal{J}^{+}, \mathcal{F}$

A delivery closes when evidence confirms its specified goal relation. A release command or a temporary empty hand alone leaves that relation unresolved. Updating one responsibility preserves its teammate's compatible transport. Closure makes successor work eligible for fresh grounding and

admission. In the recorded example, environment confirmation and the later closure record occur at distinct simulation frames. The update is associated with the evidence that supports it. It allows the reconciliation delay shown in the record.

C.7 Prompt Construction and Response Interfaces

CoELA builds prompts from the task, state, action history, dialogue history, and available actions. CoTS adds collaborative proposals, teammate critique, and plan evaluation (Zhang et al., 2024; Zu et al., 2025). ACTS retains these components and adds the responsibility and evidence fields used in continuation checks. The six templates below give normalized input–output specifications. They summarize each interface's information and response requirements. Recorded prompt excerpts follow below.

Shared context. Relevant templates identify task goal units, observation facts and their sources, the current plan, and action and dialogue history. They also distinguish immediate responsibilities from conditional future work. Evidence access follows the permitted observation setting throughout. Output fields separate intended effects from confirmed effects.

Table 8: Prompt components and their roles. Fields describe the common interface, not an additional source of privileged observations.

Component	Role
Task and goal units	Specify requested object instances or counted goal relations.
Observation evidence	Record supported facts with source and observation boundary.
Current responsibilities	Identify agent, target, stage, progress, and closure requirements.
Action/dialogue history	Retain recent execution and received task-relevant reports.
Available action descriptions	Restrict realizations to the environment adapter's interface.
Unresolved requirements	Retrieve $\mathcal{D}_t$ and rechecked $\mathcal{J}_t$; include new search requirements from $\mathcal{J}$ for revisions.
Proposed plan and feedback	Support planner–critic refinement without overwriting root evidence.

Template 1: shared-plan proposal.

```
Role: propose a collaborative continuation.
Input: TASK; GOAL UNITS; ROOT EVIDENCE; CURRENT PLAN;
CURRENT RESPONSIBILITIES; ACTIVE GAP REQUIREMENTS; RECHECKED
REJECTION REQUIREMENTS; ACTION HISTORY; DIALOGUE; AVAILABLE
ACTIONS.
Propose immediate work for each agent and any
conditional later steps. Retain compatible progressing
responsibilities. Bind each new immediate assignment to an
agent, target, and goal. State its current prerequisites
and expected effect. A future effect must remain
conditional and must not be reported as observed.
Return: joint plan; immediate assignments; conditional
successors; task-relevant message to the teammate.
```

Template 2: teammate reflection and planning value.

```
Role: evaluate an admitted collaborative proposal.
Input: PROPOSAL; ROOT EVIDENCE; RESPONSIBILITIES; DIALOGUE;
AVAILABLE ACTION DESCRIPTIONS.
Assess the division of work and movement cost using
the inherited evaluator's scoring criteria. Give a
task-relevant suggestion and its planning value. Do not
treat a predicted grasp or delivery as an observed effect.
Keep planning value separate from the admission decision.
Return: allocation assessment; movement assessment;
planning score in the inherited format; feedback message.
```

The critic estimates planning quality in the current context. This value guides selection among continuations that have passed admission.

Template 3: current-plan assessment.

```
Role: assess whether the current shared plan remains
useful.
Input: CURRENT PLAN; CURRENT RESPONSIBILITIES; RECENT
OBSERVATIONS; ACTION OUTCOMES; TASK PROGRESS; DIALOGUE.
Identify which assignments remain compatible and which
may require revision. Separate an ongoing responsibility
from one with a confirmed terminal effect. Indicate
the condition motivating a revision, without declaring a
physical effect from plan text alone.
Return: retained assignments; proposed revisions;
supporting evidence references; optional planning-level
evaluation.
```

Responsibility updates follow the executable and observed-effect contracts. An individual responsibility can close while the rest of the plan remains useful.

Template 4: unmet-condition revision.

```
Role: revise an unsupported continuation.
Input: REJECTED PROPOSAL; UNMET REQUIREMENT; RELEVANT
EVIDENCE; COMPATIBLE ONGOING WORK; AVAILABLE ACTIONS.
Change the affected assignment to address the unmet
requirement. If delivery lacks possession, propose
supported acquisition before conditional delivery. If
evidence is missing, propose a grounded inspection when
available. Preserve compatible teammate work.
Return a new candidate in the ordinary proposal format. The
revision will undergo the same admission checks.
```

Template 5: plan parsing.

```
Role: translate the selected branch into an immediate
realization.
Input: SELECTED PLAN; AGENT; GROUNDED TARGET; GOAL; CURRENT
STAGE; AVAILABLE ACTIONS; FRESH EVIDENCE.
Choose a high-level realization compatible with the
selected responsibility. Preserve the target and intended
effect. Do not substitute another goal merely to obtain
an executable command. Keep later steps conditional until
their prerequisites are supported.
Return: responsibility binding; selected available action;
expected intermediate or terminal evidence.
```

Template 6: effect and update record.

```
Input: AUTHORIZED RESPONSIBILITY; ISSUED COMMAND; EXECUTION
RECEIPT; NEW EVIDENCE; PROGRESS AND CLOSURE CONDITIONS.
Associate the outcome with the same agent and target.
Record supported intermediate progress or confirmed closure.
Distinguish release from delivery and supersession from
completion. Keep undecided status when the evidence is
insufficient.
Output record: retained/closed/superseded/failed status
as supported; evidence reference; affected task units;
requirement to check a successor.
```

This block specifies the update procedure's inputs and outputs. The checker and environment adapter establish physical relations directly. Each effect check can therefore proceed without a separate LLM call.

C.7.1 TDW PLANNING INTERFACE DETAILS

TDW planning requests retain the proposal/reflection structure of CoTS and append the execution information needed by ACTS. The planning context separates verified facts, current holding, confirmed completion, unresolved observations, text-only claims, and conflicts. Candidate actions are marked as `START`, `CONTINUE`, `PREEMPT`, or `WAIT`; progressing work normally continues, new work requires an idle/closed slot, and preemption requires an interruptible stalled or failed responsibility. The prompt also states that only environment-verified delivery counts as completion. These markers guide proposal construction, while the admission rule decides whether a branch enters search. Reflection keeps the inherited distance/task-allocation evaluation, and effect checks are performed from the execution record rather than an additional LLM call.

The normalized Templates 1–6 above are the paper specification of these interfaces. For completeness, the ACTS-specific part of the TDW proposal request has the following structure (the long inherited task instructions and examples are omitted):

```
Goal: [requested targets and counts]
Previous action plan: [current shared plan]
Dialogue history: [recent teammate messages]
Progress evidence: [verified facts; current holding;
confirmed completion; observed unresolved targets;
unverified text claims; conflicts]
Current responsibilities: [agent; action family; grounded
target; stage; interruptibility]
```

The request then asks for exactly one transition per agent and marks it as `START`, `CONTINUE`, `PREEMPT`, or `WAIT`. A progressing responsibility normally uses `CONTINUE`; `START` requires an idle or closed slot; `PREEMPT` requires a stalled/failed responsibility that may be interrupted. The instruction also states that agents may not duplicate a target and that text-only claims such as "already transported" are not completion evidence.

Reflection keeps the CoTS planner–critic role: the teammate evaluates work allocation and movement cost after the branch has passed admission. The response provides the two planning scores and task-relevant feedback the tree uses. Plan parsing receives the selected responsibility, grounded target, available actions, and fresh evidence; effect/update records receive the authorized responsibility, issued command, execution receipt, new evidence, and closure contract. These are the only prompt/interface details added here because they change the ACTS mechanism; the inherited benchmark task wording is unchanged from the CoTS-style interface.

Selected recorded constraint text. The following excerpt is retained because it specifies how language proposals refer to the persistent responsibility state before the programmatic checker runs:

```
Return a complete joint branch with exactly one transition
for Alice and one for Bob. Mark each action by appending
[ACTS_MODE=START], [ACTS_MODE=CONTINUE],
[ACTS_MODE=PREEMPT], or [ACTS_MODE=WAIT].
A progressing responsibility must normally be CONTINUE.
START requires IDLE/CLOSED. PREEMPT requires a
stalled/failed interruptible responsibility. Do not assign
the same target to both agents and do not invent object
transfer.
Do not repeat invalid branches in the current alignment
context. Unknown preconditions require evidence acquisition
or a grounded repair and are not directly executable.
```

Two shorter instructions distinguish current execution from text claims:

```
Holding agent should deliver or repair.  An empty-handed
agent should collect the next unresolved target or explore;
it must not repeat transport without held deliverables.
Use only environment-verified delivery as completed.
Text-only "already transported" claims are not completion
evidence.
```

These sentences constrain what the proposer may claim; they do not themselves establish admissibility. The candidate is parsed into agent–target–action bindings and checked against the current root evidence. A refinement request includes the rejected proposal, teammate feedback when available, the same root evidence, and the unmet requirements. The system then checks the revised proposal again rather than accepting it, since a repair prompt generated it.

Reflection and publication fields. For an admitted TDW branch, the teammate response contains a distance score, a task-allocation score, and a message explaining the suggested change. The inherited CoTS evaluator combines the two scores. The selected branch is published from the checked responsibilities; this avoids changing the grounded targets during final text formatting. Later execution still has to satisfy the corresponding possession or delivery contract before the responsibility advances or closes.

C.8 Worked Search Example

Consider distinct object instances x and y. At the fixed root, Bob holds x and is progressing toward delivery. Alice has capacity and evidence that y is available. The candidates below illustrate admission and refinement under this evidence.

Table 9: Plan refinements at a fixed execution context. Future delivery is conditional, not an observed effect.

Role	Shared continuation	Admission and evaluation
Root context	Bob holds x; Alice does not hold y.	Facts remain fixed during search.
Rejected proposal	Alice acquires Bob's x; Bob leaves delivery.	Ownership/continuity fails; no ordinary reflection.
Rejected proposal	Alice delivers y immediately; Bob continues.	Possession support fails.
Admitted revision	Alice acquires y; Bob continues delivering x.	Evaluate compatible current assignments.
Final selected plan	Retain the current assignments; deliver y only after its grasp is confirmed.	Return a plan for fresh activation, not a completed task.

Revisions change the proposed work allocation while E_t stays fixed. A rejected proposal can guide the next candidate. In Algorithm 1, only admitted candidates receive critic values and evaluated descendants. After execution confirms a grasp or delivery, Algorithm 2 reconciles the affected responsibility. The next planning call uses the updated evidence as its root. This evidence determines which continuations are admissible in that call.

Table 10: One recorded proposal–reflection–publication sequence on TDW-MAT Stuff.

Stage	Recorded content	Role in ACTS
Root	Both agents are idle; a purse is available to Alice and a pen to Bob	Fix the grounded root context and candidate bindings
Proposal	Alice acquires the purse; Bob acquires the pen	Form an admissible candidate continuation
Reflection	Distance score 3, task-allocation score 4; planning value 0.7	Evaluate and rank an admitted branch
Expansion	Three revised proposals are returned under the same root evidence	Continue search without treating predicted effects as facts
Publication	The selected branch preserves the Alice–purse and Bob–pen bindings	Dispatch checked responsibilities to execution

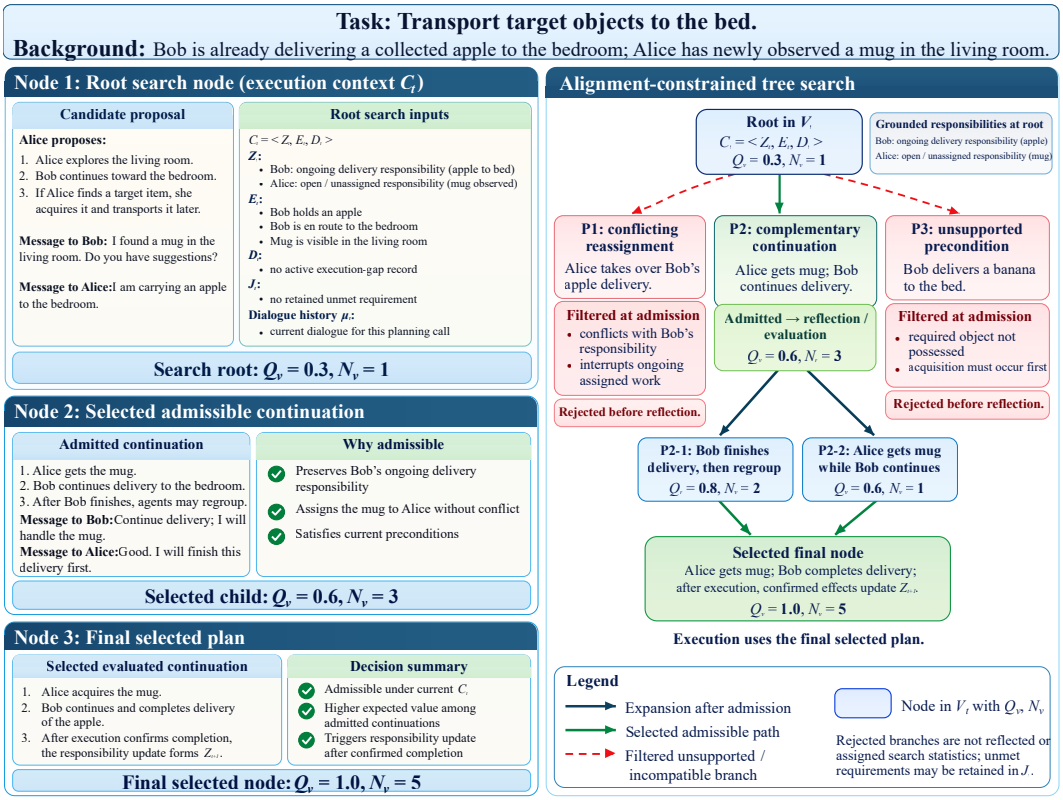
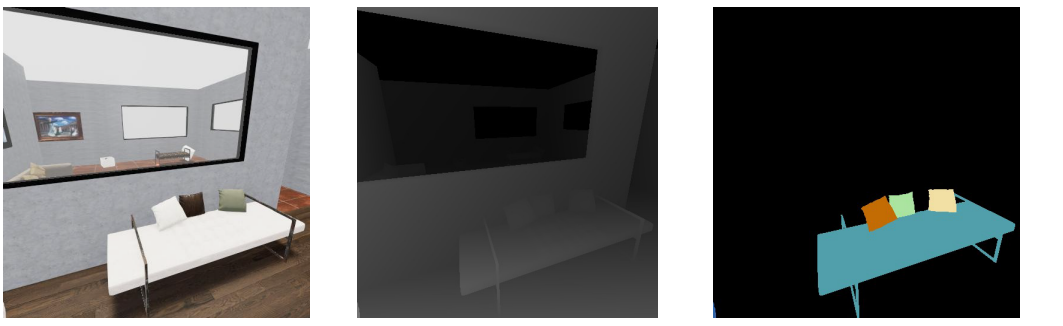


Figure 6: **Detailed ACTS search illustration.** At a fixed root execution context C_t, candidate continuations are checked before reflection and expansion. Unsupported or incompatible branches are filtered at admission, while admitted branches receive planner–critic evaluation and may be further refined. The displayed Q_v and N_v values are illustrative search statistics rather than measured execution outcomes. The selected node remains a plan; only subsequent execution evidence updates the corresponding responsibilities and forms Z_{t+1}. Algorithms 1–2 and Table 9 specify the corresponding admission, search, and update semantics.



(a) RGB **(b) Depth** **(c) Oracle Instance Annotation**

Figure 7: TDW-MAT observation channels from a common agent viewpoint: RGB, depth, and visible-instance annotation. These channels provide the local perceptual inputs used by the agent, whereas the external task evaluator operates from separate environment-level views.

C.8.1 ILLUSTRATIVE SEARCH INSTANCE

A recorded TDW-MAT Stuff search follows the same planning sequence illustrated in Figure 6. At the root, the grounded context provides two available target acquisitions under the current evidence: a purse for Alice and a pen for Bob, while both agents are otherwise idle. The joint acquisition continuation passes admission, is reflected under the same fixed root context, receives teammate-distance and task-allocation scores of 3 and 4 (planning value 0.7), and preserves the same two grounded target bindings when the selected branch is published. The full search uses five language-model requests across proposal, reflection, and expansion. As in the schematic example, physical grasp and delivery effects are not established by the planner text itself; they are confirmed only later by execution evidence.

The reflection record indicates that the assignments are feasible, while also noting that the distance arrangement could be improved. The published continuation therefore keeps the two grounded targets unchanged rather than rewriting the assignment. This recorded exchange complements the schematic tree in Figure 6: it exposes an actual planner–critic interaction on a benchmark trace, while still requiring later execution evidence to establish whether the planned acquisitions and deliveries physically occur.

D BENCHMARKS, BASELINES, AND EXPERIMENTAL PROTOCOLS

We follow the standard TDW-MAT and C-WAH task sets used by CoELA and CoTS (Zhang et al., 2024; Zu et al., 2025). This section records only the benchmark details that affect ACTS or the interpretation of the comparisons, rather than repeating the full benchmark descriptions.

D.1 BENCHMARK INTERFACES

TDW-MAT contains 24 test episodes (12 Food, 12 Stuff), ten requested target units per task, and a nominal 3,000-frame horizon (Gan et al., 2022; Zhang et al., 2024). Oracle perception annotates visible identities; the non-oracle setting uses the visual perception pipeline. Both settings provide the permitted pose, holding, communication, and action-outcome fields. Primitive motion, grasp, container interaction, release, and communication may span multiple simulation frames.

C-WAH contains ten tasks from five household activity types with a 250-step horizon (Puig et al., 2018; 2021; Zhang et al., 2024). Goals are counted as ON/IN relations. We evaluate its symbolic and visual observation modes separately. Table 13 summarizes how benchmark evidence enters the contracts; we add no evaluator-only scene graph to planning.

Table 11: Task content used by the two benchmarks.

Benchmark	Task families / targets	Relevant interaction structure
TDW-MAT Food	apple, banana, orange, bread, loaf bread, burger	acquire targets; optional container use; deliver to bed
TDW-MAT Stuff	calculator, mouse, pen, lighter, purse, iPhone	same transport structure with different objects/containers
C-WAH	afternoon tea, dishes, meal, groceries, dinner table	satisfy counted ON/IN relations with navigation, grasp, receptacle access, and placement

Table 12: C-WAH activity families and the relation to be satisfied. Episodes instantiate the requested object counts.

Activity	Goal form
Prepare afternoon tea	Requested food/drink units ON the coffee table
Wash dishes	Requested plates/forks IN the dishwasher
Prepare a meal	Requested food/drink units ON the dining table
Put groceries	Requested grocery units IN the refrigerator
Set a dinner table	Requested plates/forks ON the dining table

Environment details used by the contracts. In TDW-MAT, an agent has two hands; a held container can carry up to three targets. The reference visual observation is 512×512 RGB-D with a 90° field of view, together with the benchmark's pose, holding, and message fields. Motion, turning, grasping, putting in, releasing, and communication are primitive interactions. A high-level acquisition or delivery responsibility may therefore span several primitive commands and simulation frames. C-WAH uses the five standard activity families (afternoon tea, dish washing, meal preparation, grocery storage, and dinner-table setup). Symbolic observation exposes local object properties and relations under the benchmark scope. Visual observation uses 256×512 RGB-D with a 60° field of view and the allowed auxiliary fields. The relevant interactions are navigation, turning, grasp, opening/closing receptacles, placement, and communication. A counted goal, such as two plates on a table, requires distinct object instances; ACTS therefore keeps the object binding until the corresponding ON/IN relation is confirmed.

Table 13: Benchmark information used by ACTS.

Setting	Evidence used online	Typical closure
TDW-MAT	Visible/recognized target identity, holding, pose, action receipt, delivery evidence	Requested target delivered
C-WAH symbolic	Local symbolic relations, holding, receptacle state, messages	Requested ON/IN relation
C-WAH visual	Permitted RGB-D-derived facts and auxiliary fields	Requested relation supported after execution

D.2 Baselines and Numerical Provenance

Our comparisons cover hierarchical planning, multi-agent sequence modeling, and recent LLM-based embodied cooperation. RHP/CoRHP and MHP/CoMHP are the hierarchical planning references adopted from CoELA, while MAT provides the centralized multi-agent sequence-modeling baseline in TDW-MAT (Wen et al., 2022; Zhang et al., 2024). The LLM-based methods include CoELA, RoCo, CaPo, CoTS, PCE, and ACTS (Zhang et al., 2024; Mandi et al., 2024; Liu et al., 2025; Zu et al., 2025; Seo et al., 2026). REVECA provides an additional reference for adaptive planning and trajectory validation (Seo et al., 2025).

RHP combines heuristic transport decisions with map-based navigation and frontier exploration, with CoRHP extending it to cooperative execution. MHP uses high-level MCTS with regression-planning execution on C-WAH, and CoMHP provides its cooperative counterpart. MAT models multi-agent decision making as sequence prediction and follows its reported oracle-perception TDW setting (Wen et al., 2022; Zhang et al., 2024). Among language-based methods, CoELA provides the modular LLM-agent foundation, RoCo coordinates agents through dialogue, CaPo maintains and updates a cooperative meta-plan, CoTS searches planner–critic refinements, and PCE reasons explicitly about uncertainty in planning assumptions (Zhang et al., 2024; Mandi et al., 2024; Liu et al., 2025; Zu et al., 2025; Seo et al., 2026). Together, these baselines span complementary approaches to task allocation, communication, collaborative planning, search, and uncertainty-aware reasoning.

CoTS-S retains the CoTS planner while using the executor shared with ACTS.

D.3 Repeated Runs and Aggregation

Locally evaluated benchmark methods use three independent runs with matched task seeds across methods. Tables 1 and 2 report mean performance across these runs, using the same task identities and benchmark metrics within each run.

D.4 Online Control Conditions

The controls modify different stages of the same planning–execution stack. Table 15 summarizes the intervention applied in each condition. The online controls vary planning context, admission place-

Table 14: Numerical sources for the benchmark tables.

Method	Benchmark	Source
RHP/CoRHP; MHP/CoMHP; MAT; CoELA	As applicable	CoELA publication
RoCo	Both	CoTS comparison
CaPo	Both	Our evaluation
PCE	Both, GPT-4	Our matched evaluation
CoTS	Both	Our reproduction
CoTS-S	Both	Shared-executor control
ACTS	Both	Our evaluation

ment, or update behavior, whereas the offline projections in Appendix D.8 vary only the information available for fixed-context assessment.

Table 15: Online control conditions. The raw tree gate isolates admission placement; full ACTS additionally revises proposals from unmet conditions and updates responsibilities after execution.

Variant	Planning context / proposal source	Admission placement	Update/recovery distinction
Original CoTS	Original CoTS context/search	None	Original plan/progress update
CoTS-S	Original CoTS search; shared executor	None	Original plan update
Evidence-text	Execution facts verbalized in context	None	Context only; no ACTS admission
Structured context	Same facts exposed as fields	None	Context only; no hard tree constraint
Raw post-search guard	Ordinary proposals + shared recovery	After reflection	Fresh activation/update retained
Raw tree gate	Ordinary proposals + shared recovery	Before reflection/expansion	Isolates pre-reflection admission placement
Without unmet-condition revision	ACTS condition with unmet-condition revision disabled	ACTS checks retained	Other ACTS settings retained
Full ACTS	Ordinary + unmet-condition revisions/recovery	Before reflection + fresh activation	Preserve/close/supersede responsibilities; recheck $\mathcal{J}$

D.5 Implementation and Model Configuration

Experiments use an Intel Core i9-14900K CPU and an NVIDIA GeForce RTX 4090 GPU. Benchmark comparisons use GPT-4 and GPT-3.5-Turbo as indicated, while mechanism analyses use GPT-4. The decoding configuration uses temperature 0.7, top-p 1, and a maximum output length of 512 tokens.

Table 16: Experimental settings for the TDW mechanism analyses.

Setting	Value
Runtime / agents	TDW 1.11.23 / two language-model agents
Candidate setting / maximum tree height	3 / 2
Regeneration rounds / reflection retries	2 / 2
Delivery-evidence wait / no-progress window	3 / 20 interaction steps
Independent benchmark runs	3 per locally evaluated method; matched task seeds
Prompt interfaces	Templates in Appendix C.7

D.6 Aggregation and Relative Changes

For locally evaluated TDW methods, Avg. is the arithmetic mean of the reported Stuff and Food values, while published aggregates are retained as originally reported. Relative change from CoELA is computed as $100(x-b)/b$ for TR and $100(b-x)/b$ for C-WAH completion steps. The main text reports absolute percentage-point or step differences. Best and second-best markings are determined from distinct LLM-based results within each row, with ties preserved.

D.7 Proposal–Execution Mismatch Audit

To characterize proposal–execution mismatch, we audit a TDW-MAT Stuff execution under oracle perception using the raw post-search guard. After deduplication by search boundary and semantic candidate identity, the audit includes 145 ordinary reflected proposals; we analyze 33 recovery proposals separately. Each proposal is paired with the grounded root state available before reflection and evaluated using holding state, active responsibility and lifecycle, interruptibility, target and room bindings, confirmed completion, cleanup obligations, and the immediate joint step. Future observations and final episode outcomes are not used.

Each proposal receives one mutually exclusive primary label. Among the 145 proposals, 38 lack support for a task precondition or grounding, 16 replace work that is still progressing, six leave required cleanup unresolved, and 19 lack sufficient current execution evidence. The remaining 66 support the immediate action, yielding the distribution in Table 3. Reclassifying the 38 precondition/grounding cases as insufficient-evidence cases rather than unsupported cases leaves the aggregate number lacking sufficient support unchanged at 79/145.

A fixed-seed manual consistency check re-evaluates 30 sampled root/proposal pairs without access to future outcomes and reproduces the original primary label in all 30 cases. The eligibility checker flags 143/145 proposals under its broader criterion, which additionally evaluates later steps, formatting, mode transitions, and implementation constraints.

D.8 Fixed-Context Assessment

The fixed-context analysis uses 24 contexts sampled with a fixed random seed from 434 deduplicated candidate-assessment contexts collected from two Stuff executions. Candidate identity is held fixed across six information conditions. A blinded GPT-4-assisted assessor receives the task, proposed immediate joint step, and the corresponding evidence projection, while method outputs, scores, admissible-action lists, checker decisions, and future outcomes are withheld. All assessments are performed offline on the fixed contexts.

Coverage for information condition c is

$$\mathrm{Coverage}(c) = \frac{N_{\mathrm{admissible}}(c) + N_{\mathrm{inadmissible}}(c)}{24}. \tag{19}$$

An `unknown` label indicates that the available evidence is insufficient for a decision. Since these contexts have no independent gold continuation label, coverage measures decision determinacy rather than classification accuracy.

Information projections. The six conditions are: plan/task text only; the complete execution record serialized as text; a complete structured record; and structured records with ownership, progress history, or closure evidence withheld. Duplicate encodings of each withheld fact group are removed before request construction, while candidate wording remains fixed. This isolates the contribution of the corresponding execution information.

Assessor instruction and outputs. The assessor evaluates only the immediate joint step from evidence available at that decision: `admissible` when the evidence supports the step, `inadmissible` when it contradicts the step, and `unknown` when the evidence is insufficient. Predicted future effects are not treated as current facts, and only previously observed objects may be used. Each condition is evaluated once in a fresh request at temperature 0.7 with a 512-token limit and JSON output. The response contains `label`, `reason`, `evidence_paths`, and `missing_evidence`; request or parsing failures are recorded separately.

Example format. One full-context example contains the immediate branch

```
Alice:  transport objects I am holding to the bed
[MODE=CONTINUE].
Bob:  go to a target room [MODE=START].
```

The evidence projection states that Alice holds the task iPhone assigned to her transport responsibility and that this execution phase is `PROGRESSING`; Bob has no active target responsibility and

his previous phase is `CLOSED`. The phone is absent from the confirmed-delivery list. The resulting assessor record is

```
{
  "label":  "admissible",
  "reason":  "Alice continues the transport responsibility
supported by her holding state; Bob's new navigation does
not conflict with an active responsibility.",
  "evidence_paths":  [observation.holding,
ownership.responsibilities, progress.execution],
  "missing_evidence":  []
}
```

Identifiers are shortened for display. The example illustrates how a definite continuation judgment is grounded in the evidence fields exposed under the corresponding information condition.

D.9 Offline Assessment of the Formal Gaps

Equation (4) defines the four gap indicators. The offline estimate $\widehat{d}^i_{t,\tau}$ associates each formal quantity $d^i_{t,\tau}$ with its verification boundary and the execution evidence available at that phase.

Let $m^i_{t,\tau}$ indicate whether the trace contains the relevant verification boundary and evidence, and let $r^i_{t,\tau}$ denote the reconstructed relation assessment, taking values true, false, or $\mathsf{unknown}$. The offline estimate is

$$\widehat{d}^i_{t,\tau} = \begin{cases} 0, & m^i_{t,\tau} = 1, \quad r^i_{t,\tau} = \mathsf{true}, \\ 1, & m^i_{t,\tau} = 1, \quad r^i_{t,\tau} = \mathsf{false}, \\ \mathrm{NA}, & \text{otherwise.} \end{cases} \tag{20}$$

Realization assessment matches each responsibility to its agent, grounded target, issued command, and execution receipt. Effect assessment applies the corresponding progress or closure condition. A gap is resolved when the affected relation is restored, or the responsibility is validly retired. These trace-based quantities connect the formal interfaces to their corresponding execution records.

D.10 Task, Candidate, and Cost Measures

Task outcome. For TDW-MAT, $\mathrm{TR} = 100 N_H / K$, where N_H is the number of requested units delivered by the evaluation horizon and K is the requested total. The benchmark's object-level goal evaluation supplies N_H.

C-WAH reports average completion steps separately for visual and symbolic observations. ACTS completes all ten tasks within the 250-step horizon in both settings (10/10; 100% success). For each method and observation setting, the reported value is the arithmetic mean over the ten benchmark tasks.

Candidate evaluation. We deduplicate reflected candidates by search identity and semantic candidate signature. The placement comparison evaluates the ordinary raw-proposal stream, with recovery-generated candidates accounted for separately. For the assessed reflected pool,

$$\mathrm{InvalidEval} = 100 \, \frac{N_{\text{reflected, checker invalid}}}{N_{\text{reflected, assessed}}}. \tag{21}$$

The eligibility checker supplies the judgment. This measure quantifies the fraction of ordinary reflected candidates that fail the eligibility check under each placement condition.

Admission and closure. A search with new admission contains at least one newly admitted continuation; retained executable responsibilities are counted separately.

A matched completion–closure pair associates an environment-confirmed effect with the corresponding responsibility-closure record, linking physical task progress to its responsibility update.

Planning usage. Requests count language-model invocations, and total tokens sum input and output tokens. Physical execution time is measured separately in environment frames.

D.11 Comparison Units

Task-category tables summarize benchmark performance over the stated task sets and, for locally evaluated methods, over three independent runs with matched task seeds. Mechanism analyses use individual executions or matched execution pairs, depending on the corresponding protocol. The unmet-condition revision ablation contains one matched Food pair and one matched Stuff pair, with both configurations sharing the same task and seed within each pair. The 63 searches and eight matched completion–closure events reported in Section 5.4 aggregate the two complete ACTS executions from these pairs.

Table 18 reports a separate representative Stuff execution. Fixed-context analyses compare identical candidate contexts across information conditions. Candidate-level statistics use deduplicated search–candidate pairs, while responsibility analyses use matched execution and closure events. These units correspond to different stages of the ACTS planning–execution process and are reported at their respective analysis levels.

E Additional Results and Execution Cases

This section provides the execution-level records underlying the mechanism analyses in the main text, including representative outcomes, control variants, field ablations, planning cost, and responsibility-update cases.

E.1 Representative Execution Outcomes

Table 17: **Representative TDW-MAT execution outcomes.** Each row reports one complete execution under the indicated method and perception setting.

Task	Method	Perception	TR	Frames
Stuff	ACTS	Oracle	100% (10/10)	2866
Food	ACTS	Oracle	100% (10/10)	2915
Stuff	ACTS	Oracle	100% (10/10)	2589
Food	ACTS	Oracle	100% (10/10)	2310
Stuff	CoTS-S	Non-oracle	70% (7/10)	3000
Food	CoTS-S	Non-oracle	60% (6/10)	3000
Stuff	CoTS-S	Non-oracle	80% (8/10)	3000
Food	CoTS-S	Non-oracle	90% (9/10)	3000

These records provide execution-level anchors for the aggregate benchmark results. The ACTS oracle runs shown here complete all ten target objects with different completion times, while the CoTS-S non-oracle executions illustrate the corresponding horizon-limited outcomes under visual perception. The following analyses use selected executions from these records to examine search and update behavior in greater detail.

E.2 Recorded Outcomes of the Comparison Variants

Original CoTS and the ACTS reference both reach 100% TR in this execution, while ACTS finishes earlier and uses fewer language-model requests. The raw tree gate reaches 50% despite eliminating checker-invalid ordinary proposals from reflection, showing that admission placement interacts with proposal revision and responsibility reconciliation in determining execution performance.

E.3 Execution-Context Field Ablation

All six conditions evaluate the same 24 candidate contexts. Appendix D.8 specifies the corresponding information projections and blinded assessment protocol. The field removals show the largest

Table 18: **Representative oracle-perception TDW-MAT Stuff comparison.** The raw tree gate isolates admission placement, while the remaining variants modify context, admission placement, or the complete ACTS mechanism.

Variant	TR	Final frame	Requests
Original CoTS	100%	2988	219
CoTS-S	80%	3001	210
Evidence-text	90%	3002	295
Structured context	90%	3001	259
Raw post-search guard	70%	3004	254
Raw tree gate	50%	3002	319
ACTS reference	100%	2589	146

Table 19: **Execution-context field ablation on 24 fixed candidate contexts.** Labels are produced by the blinded model-assisted assessor. Coverage is the fraction receiving a non-`unknown` judgment.

Information available	Admissible	Inadmissible	Unknown	Coverage
Plan and task description	0	1	23	4.2%
Full execution record as text	7	16	1	95.8%
Structured record, no ownership	5	12	7	70.8%
Structured record, no progress history	6	14	4	83.3%
Structured record, no closure evidence	5	14	5	79.2%
Full structured record	2	19	3	87.5%

coverage reduction when ownership information is removed, with progress history and closure evidence also contributing to definite continuation judgments.

E.4 Paired Execution Efficiency and Planning Usage

Figure 4 visualizes completion frames and request counts for the matched Food and Stuff pairs. Table 20 complements the figure with total token usage.

Table 20: **Planning cost for the two matched TDW-MAT pairs.** All four executions complete 10/10 target objects.

Task	Variant	Frames	Requests	Total tokens
Food	Without proposal revision from unmet conditions	2897	196	979,214
Food	ACTS	2310	182	960,962
Stuff	Without proposal revision from unmet conditions	2613	108	572,759
Stuff	ACTS	2589	146	825,926

On Food, ACTS reduces completion frames, requests, and total token usage. On Stuff, it produces a smaller reduction in completion frames while increasing planning usage. The two matched pairs therefore show that unmet-condition revision can improve physical execution while its planning cost varies with the task.

E.5 Admission Placement and Candidate Accounting

Figure 3 compares the same eligibility criterion at two locations in the search procedure. The raw post-search guard reflects 145 ordinary candidates, of which 143 are checker-invalid, whereas the raw tree gate reflects ten ordinary candidates and none are checker-invalid. Shared recovery contributes 33 and 51 additional reflected candidates, respectively. Moving the same check before reflection therefore changes which ordinary branches consume planner–critic evaluation while leaving the eligibility criterion itself unchanged.

E.6 Responsibility Maintenance and Recorded Case

Across the complete Food and Stuff ACTS paired executions, 62 of 63 searches admit at least one new continuation, and eight environment-confirmed completion events are matched to the responsibility closure that follows them. These counts aggregate the two complete ACTS executions used in the paired analysis.

Table 21: **Admission and closure events in the two complete ACTS paired executions.**

Quantity	Count
Recorded searches	63
Searches with at least one new admission	62
Searches without a new admission	1
Matched environment-completion / responsibility-closure pairs	8

Figure 5(a) follows one Stuff execution in detail. Agent 0 is responsible for a pen and an iPhone (y_1, y_2), while Agent 1 transports a mouse and a purse (x_1, x_2).

Table 22: **Responsibility events in Figure 5(a).**

Frame	Responsibility	Execution state and response
309	Agent 1, x_1, x_2	Transport PROGRESSING; retain ongoing work.
309	Agent 0, y_1, y_2	Separate delivery responsibility PREPARED.
318	Agent 0, y_1, y_2	Delivery confirmed; completed units increase by $+2$.
386	Agent 0, y_1, y_2	Responsibility CLOSED; refresh remaining assignments.

Agent 1 still holds its targets after Agent 0's delivery is confirmed. The subsequent refresh therefore closes Agent 0's completed responsibility while preserving Agent 1's progressing transport, illustrating responsibility-specific reconciliation within a shared collaborative branch.

E.7 Environment-Confirmed Progress in Figure 5(b)

Panel (b) uses separate Food and Stuff executions from the responsibility case above. The checkpoints below provide the exact evaluator values underlying the visualized task progress.

Table 23: **Environment-confirmed task progress shown in Figure 5(b).**

Task	Checkpoint	Frame	Completed
Food	Earlier	425	1/10
Food	Later	474	2/10
Stuff	Earlier	302	1/10
Stuff	Later	363	2/10

The evaluator's object-level goal record supplies these counts, while the agents continue to plan from the permitted online observations. In both examples, the later checkpoint records one additional environment-confirmed goal unit.

E.8 Execution-Level Findings

The execution records expose three properties of ACTS at the boundary between collaborative planning and ongoing work. First, a responsibility can persist across multiple primitive actions and planning calls while retaining the same grounded assignment. Second, environment confirmation and responsibility closure may occur at different simulation frames because reconciliation follows incorporation of the corresponding execution evidence. Third, closing one agent's completed responsibility preserves a teammate's compatible progressing responsibility.

These properties also clarify the role of evidence status in ACTS. Missing observations can leave a required relation *unknown*, delaying activation until sufficient evidence becomes available, while

an issued command alone does not establish its intended physical effect. Responsibilities therefore advance, close, fail, or become superseded according to their execution contracts and observed evidence rather than plan text alone.

TDW-MAT and C-WAH use the same search and responsibility-update procedure with benchmark-specific observation and action semantics. Their contracts instantiate different target bindings, action families, progress conditions, and closure predicates. Extending ACTS to additional environments therefore requires corresponding execution-contract bindings; learning these bindings automatically, scaling to larger teams, and evaluating under noisy real-robot observations remain directions for future work.