WHEN DOES NEGOTIATION HELP? BENCHMARKING EMBODIED MULTI-AGENT COLLABORATION UNDER SUCCESSIVE TASK REVISIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Successive task revisions require embodied teams to accommodate new requirements during execution, and performance losses may reflect both final-task expansion and online adaptation. Existing evaluations lack upfront and online conditions with matched final requirements, making these sources difficult to distinguish. We introduce a controlled benchmark of 50 evaluation tasks for LLM-driven heterogeneous teams, synchronizing task and evaluator revisions during continuous physical execution and providing both conditions with matched intended final requirements. Model-equal averages across nine backbones show that 78.8% of the success-rate loss from original to online-revised tasks is already present when final requirements are provided upfront, while online introduction is associated with an additional loss of 5.67%. This online gap occurs on every backbone and is larger in object-state and spatial-adjacency tasks; tasks with temporal constraints exhibit the largest expansion-related loss but no larger online gap. Using explicit allocation dialogue as a coordination probe across four backbones, we find negotiation gains similarly concentrated in the former two groups; broader reallocation does not necessarily improve completion. Two Dynamic configurations that select negotiation modes based on experience achieve higher success rates than full negotiation on all four backbones, each with a model-equal mean gain of 4.38 pp and lower mean action-planning token usage. These findings show that negotiation benefits are context-dependent and cannot be explained by overall task difficulty alone; the compatibility of new requirements with existing allocations and goal dependencies offers a perspective for interpreting these conditional benefits. Code and data are available at https://www.negobench.top/.

1 INTRODUCTION

Task requirements in embodied collaboration may evolve as execution progresses. For example, a human agent and a Spot robot are tidying objects when the user asks them to clean an object that has already been moved, adds new tidying goals, and specifies the order of subsequent operations. These requests change what constitutes task completion without undoing the physical consequences of earlier actions: objects remain where they have been moved, agents may be holding or manipulating objects, and teammates have accumulated different local observations. Performance degradation under task revisions can arise from intertwined sources. On the one hand, the final task may involve additional goals, more complex object-state requirements, or stronger temporal constraints, making it harder than the original task even when fully specified from the outset. On the other hand, when requirements are introduced incrementally during execution, earlier actions cannot fully account for subsequent goals, potentially increasing the need for plan adjustments, repeated operations, or renewed coordination. Because revisions differ in how they affect existing allocations and execution dependencies, controlled comparisons are needed to establish when explicit negotiation is warranted and how broadly existing assignments should be reconsidered.

Habitat 3.0 and PARTNR provide infrastructure and benchmarks for embodied collaboration involving spatial, temporal, and heterogeneous-capability requirements (Puig et al., 2024; Chang et al., 2025). Language-based approaches have explored information exchange, shared plans, and task allocation (Zhang et al., 2024; Mandi et al., 2024). These methods adapt execution plans under given

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.

task goals; their evaluations do not examine the adaptation burdens and coordination demands arising from repeated within-episode revisions to task specifications. As illustrated in Figure 1(a,b), typical fixed-task settings consider single-agent decision-making and multi-agent collaboration, respectively: the environment evolves through actions, while task completion criteria remain unchanged during execution. We study the online task revisions illustrated in Figure 1(c), where the team's task specification evolves from $\mathcal{G}_0$ to $\mathcal{G}_1, \mathcal{G}_2, \ldots$. Existing requirements unaffected by a revision remain in force, while physical execution and local observation histories continue without resets.

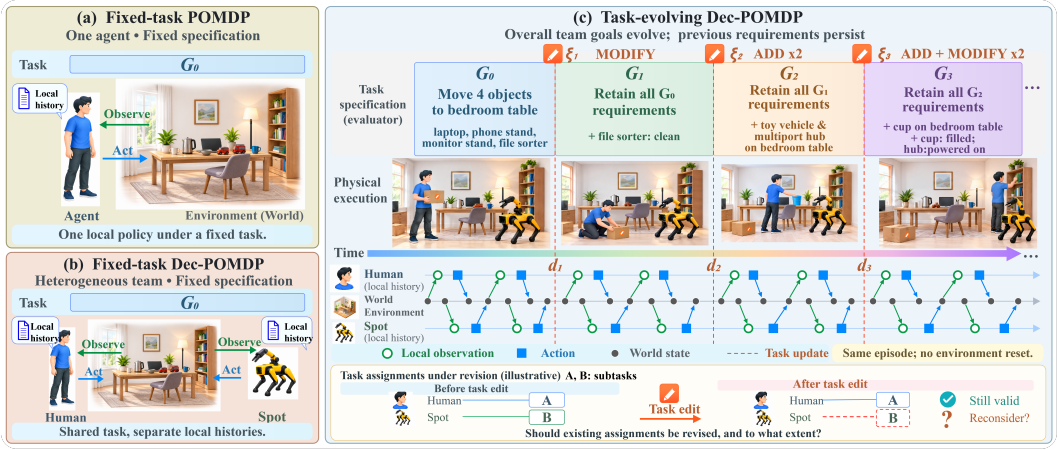


Figure 1: **Task evolution and assignment revision.** (a,b) Fixed-task single-agent and heterogeneous multi-agent settings. (c) Repeated task updates within one episode without resets.

To study this setting, we construct a controlled benchmark for embodied collaboration under online task revisions. Each task is defined by a natural-language instruction, goal propositions, and evaluation constraints, and is paired with a replayable revision sequence. Revisions are triggered by execution progress and synchronously update the task instruction and executable evaluator, while preserving physical continuity and maintaining evaluation histories consistent with the active task version. Complete revision sequences undergo semantic checks and validation through execution in the environment. The benchmark provides three conditions that start from the same initial environment state: ORIGINAL-STATIC provides the complete original task requirements at episode start; FINAL-UPFRONT provides the complete requirements of the final revised task at episode start; and ONLINE-REVISED introduces revisions according to the stored sequence during ongoing execution, matching the intended final task specification of FINAL-UPFRONT. This design allows us to examine separately the burdens associated with final-task expansion and the online revision process. We further use explicit allocation dialogue as a coordination probe, examining task completion and execution effort under different task conditions. Fixed negotiation protocols permit different scopes of allocation revision, while the experience-based DYNAMIC strategy selects a negotiation mode according to the current context. All protocols receive the currently valid task instruction, and the no-negotiation baseline likewise continues planning under updated requirements. By relating task structure and revision-induced changes to protocol outcomes, we examine how negotiation benefits align with online adaptation burden.

We focus on three questions: RQ1: How much of the completion loss and execution burden is associated with final-task expansion, and how much with online revision? RQ2: How does the additional burden of online revision vary with task structure and revision-induced changes in goals, state requirements, and dependencies? RQ3: Under which revision conditions does explicit allocation dialogue help, and how do its benefits relate to online adaptation burden and overall task difficulty? Our contributions are: (1) A controlled benchmark for online task revisions. We construct an executable revision suite comprising 50 evaluation tasks, combining continuous physical execution, synchronized task and evaluator updates, history-aware evaluation, and complete-sequence validation, with upfront and online conditions that match the intended final requirements. (2) Source diagnosis across models. Across models, we distinguish performance losses associated with final-task expansion from the residual online adaptation burden under matched final requirements, and analyze the task structures and revision conditions where this burden is concentrated. (3) Coordination di-

agnosis through allocation dialogue. We use explicit allocation dialogue as a coordination probe to test, across task conditions, whether negotiation benefits align with online adaptation burden rather than being explained by overall task difficulty alone.

2 Related Work

Embodied multi-agent evaluation. Existing benchmarks characterize embodied teamwork through goal understanding, collaborative execution, and constraint satisfaction. Watch-And-Help evaluates agents' ability to infer a partner's goal from demonstrations and assist with household tasks (Puig et al., 2021); TDW-MAT examines collaborative transport under partial observability (Zhang et al., 2024), while MAP-THOR evaluates long-horizon collaborative planning across instruction ambiguity levels and room layouts (Nayak et al., 2024). PARTNR builds on Habitat 3.0 to ground household instructions in evaluable tasks with spatial, temporal, and heterogeneous capability requirements (Puig et al., 2024; Chang et al., 2025). Communicative PARTNR examines communication range limitations and feedback granularity, while PARTNR-Dialog incorporates explicit dialogue into collaborative evaluation (Jeon et al., 2025; Zhou et al., 2026). Task specifications in these household collaboration benchmarks remain fixed within each evaluation episode. Environmental dynamics introduce another challenge: HAZARD evaluates embodied decision-making under fire, flood, and wind (Zhou et al., 2024). This benchmark concerns goal achievement under environmental change, whereas we study successive task revisions that alter goals and constraints during execution. These existing benchmark settings do not jointly cover within-episode revisions to task specifications, consistent evaluation across task versions, and an upfront control with matched final requirements.

Online task change and plan repair. Classical plan repair studies plan reuse and modification after goal or state changes, examining trade-offs among computational efficiency, plan quality, and stability (van der Krogt & de Weerdt, 2005; Fox et al., 2006). Among language-model-driven robotic systems, KGLAMP repairs environmental knowledge and replans in response to new observations and execution failures, while DRAMA reallocates work as team members join, leave, fail, and recover (Shek et al., 2026; Zhao et al., 2026). More directly addressing changing user requirements, HERON updates human–robot schedules in response to instructions that add, remove, or modify subtasks (Kim et al., 2026). On the evaluation side, InterruptBench introduces requirement additions, revisions, and retractions during long-horizon web navigation without resetting the environment, examining successive updates, interruption timing, and recovery costs (Zou et al., 2026). Its no-update reference receives only the initial instruction but is evaluated against the final intent, providing a baseline for assessing agents' ability to incorporate user updates. These studies examine plan adjustment and execution recovery under changing conditions. We complement this line of work by comparing Original-Static, Final-Upfront, and Online-Revised execution to distinguish the burden associated with final-task expansion from the additional burden of online adaptation.

Communication and coordination. Existing work supports embodied teamwork through shared plans, selective communication, and experience-based coordination. CoELA integrates communication with perception, memory, and planning, while RoCo uses multi-round dialogue to coordinate subtasks and motion planning (Zhang et al., 2024; Mandi et al., 2024). For plan maintenance, CaPo updates a shared meta-plan as execution progresses, and CoTS evaluates existing plans to determine when revision is warranted, avoiding overly frequent updates (Liu et al., 2025; Zu et al., 2025). For communication decisions, CoBel-World reasons about teammates' beliefs and intentions to select when to communicate, while CommCP uses conformal prediction to calibrate message confidence and select information for sharing (Wang et al., 2025; Zhang et al., 2026). Coordination can also draw on offline experience: Tact distills cooperative preferences into retrievable contextual rules that support decentralized execution without online messages, while LLawCo extracts cooperation laws from failure trajectories and incorporates them into agent training (Pei et al., 2026; Zhou et al., 2026). However, improvements in local coordination do not necessarily translate into task-level gains: a dialogue-based diagnosis on PARTNR finds that fewer action conflicts do not yield higher success rates in the evaluated configurations (Dongre & Hakkani-Tür, 2026). We examine when negotiation helps during successive task revisions. With all protocols receiving task updates, we use explicit allocation dialogue as a coordination probe, compare different scopes of allocation revision, and analyze negotiation benefits and execution-planning effort across task structures.

3 A CONTROLLED BENCHMARK FOR ONLINE TASK REVISION

3.1 TASK REPRESENTATION AND ONLINE REVISION

Building on Habitat 3.0 and PARTNR (Puig et al., 2024; Chang et al., 2025), we represent task version i as $\mathcal{G}_i = (I_i, \Phi_i, \mathcal{C}_i)$, where I_i is the natural-language instruction, Φ_i is an ordered list of goal propositions, and $\mathcal{C}_i$ comprises proposition dependencies and constraints on temporal order, terminal satisfaction, and argument consistency. Revisions jointly update the instruction and executable evaluation specification, while unaffected requirements remain in force. Let ξ_i denote the ith revision, d_i its application time, $j(t)$ the number of revisions applied by time t, and s_t the physical state. Then

$$\mathcal{G}_i = \xi_i(\mathcal{G}_{i-1}), \qquad G_t = \mathcal{G}_{j(t)}, \qquad s_{d_i^+} = s_{d_i^-}. \tag{1}$$

The team consists of a simulated human agent and a Spot robot, both controlled by language models. Both can transport objects, whereas state-changing operations such as cleaning and filling are available only to the human agent. Each agent selects actions based on its own observations, action results, received instructions, and communication messages, without access to the global physical state, internal evaluation history, or future revisions. After revisions are applied, both agents receive the updated cumulative instruction; an agent executing a skill uses it when it next plans. Revisions neither interrupt ongoing skills nor change existing object possession.

3.2 CONSTRUCTING EXECUTABLE REVISION SEQUENCES

The evaluation suite comprises a stratified sample of 50 tasks from the PARTNR validation split, covering 12 scenes. A separate set of 100 training tasks, disjoint from the evaluation tasks, is used exclusively to construct retrieval experience for DYNAMIC. Each task has a preconstructed revision recipe $\Xi = ((\rho_1, \xi_1), \ldots, (\rho_K, \xi_K))$, where $\rho_i \in [0, 1]$ is a completion threshold and $K \in \{0, 1, 2, 3\}$. Each revision event ξ_i may combine multiple atomic operations. Four operator families modify goal composition, object-state requirements, and execution dependencies: ADD introduces a location requirement, DELETE cancels a requirement, MODIFY adds a state requirement for an existing task object, and REORDER introduces two ordered placement goals.

Revisions are checked at eligible high-level-action boundaries, when at least one agent has no skill in progress, including at initialization. If current completion reaches the next event's threshold, the revision is applied and completion is recomputed under the updated specification before checking subsequent events. Events applied consecutively at the same boundary form a revision batch. Models and protocols use the same revision content, order, and thresholds, while application times and triggered-event counts depend on the execution trajectory. All retained sequences undergo structural checks, independent semantic checks by two language models, and human review. Feasibility is validated through manual execution in the simulator under both successive revisions and upfront final requirements. Construction procedures, validation methods, and coverage statistics are detailed in Appendix A.

3.3 MATCHED CONTROLS AND EVALUATION

Applying the complete revision sequence to the original task yields the final specification $\mathcal{G}_K$ and defines the three conditions in Table 1. All conditions share the initial physical state, agent capabilities, observation interfaces, and execution budget. The contrast between ORIGINAL-STATIC and FINAL-UPFRONT characterizes performance changes associated with final-task expansion. FINAL-UPFRONT and ONLINE-REVISED match intended final requirements, comparing completion and execution effort when those requirements are provided upfront or introduced online.

Table 1: Task conditions in the matched disclosure design.

Condition	Initial specification	Task updates	Intended final specification
ORIGINAL-STATIC	$\mathcal{G}_0$	None	$\mathcal{G}_0$
FINAL-UPFRONT	$\mathcal{G}_K$	None	$\mathcal{G}_K$
ONLINE-REVISED	$\mathcal{G}_0$	Progress-triggered revisions	$\mathcal{G}_K$

Evaluation histories are updated jointly with task specifications: retained propositions preserve their satisfaction records, new propositions receive no satisfaction timestamp predating their introduction, and cancelled requirements are removed from evaluation. The current physical state may immediately satisfy a new proposition upon introduction, without backdating its satisfaction record; retained histories continue to inform the relevant constraints. We report percent complete (PC), success rate (SR), and execution effort. PC is the fraction of active propositions that have been recorded as satisfied and pass their associated constraints; SR is the fraction of episodes in which all active requirements pass evaluation at termination. All conditions use a budget of 20,000 simulator steps. Automatic successful termination requires all scheduled revisions to have been applied and the active task to succeed. Budget-limited episodes are scored under the active task version; in ONLINE-REVISED, this may be an intermediate specification. Execution effort is measured per episode by simulator steps, mean planning calls per agent, and the sum of both agents' action-planning input and output tokens, excluding negotiation dialogue usage. The task process and evaluation definitions are detailed in Appendix B.

4 DIAGNOSING THE COST OF ONLINE TASK REVISION

We evaluate nine backbones on the same 50 tasks under Original-Static, Final-Upfront, and Online-Revised, using the NEVER protocol throughout. All conditions provide the active task instruction, which agents use to continue decentralized planning.

4.1 FINAL-TASK EXPANSION AND ADDITIONAL ONLINE BURDEN

Most of the original-to-online completion gap is already present when the final requirements are available upfront. Mean SR decreases from 73.11% under Original-Static to 46.33% under Online-Revised, a gap of 26.78 percentage points. Of this gap, 21.11 points persist under Final-Upfront, accounting for 78.8% of the total; the remaining 5.67 points constitute the additional online gap under matched intended final requirements. All nine backbones show lower SR under Online-Revised than under Final-Upfront (Table 2).

Task expansion can require cleaning, filling, or power-state operations restricted to the human agent, limiting work allocation; cross-object ordering requirements further constrain parallel execution. The substantial completion loss under Final-Upfront is consistent with the difficulty of jointly scheduling these operations even when all requirements are known initially. PC declines less than SR as tasks expand, and their separation is already evident under upfront disclosure. Location, state, and inter-object relations are not independent units of work: one action can satisfy multiple propositions. High PC reflects the preservation of local progress, whereas complete success depends on satisfying the relevant conditions jointly.

Table 2: **Three-condition completion comparison under NEVER.** A, B, and C denote Original-Static, Final-Upfront, and Online-Revised. SR and PC are percentages; SR differences are in percentage points (pp), with positive values indicating lower completion in the second condition. The macro row weights the nine backbones equally. Differences are computed before display rounding.

Model	SR (%) A→B→C	PC (%) A→B→C	ΔSR (pp) A−B	ΔSR (pp) A−C	ΔSR (pp) B−C
GPT-4o mini	72.00 → 31.00 → 28.00	87.31 → 76.67 → 72.17	41.000	44.000	3.000
DeepSeek-v4-flash	82.00 → 63.00 → 61.00	92.84 → 90.06 → 89.37	19.000	21.000	2.000
GPT-OSS-20B	81.00 → 58.00 → 55.00	90.48 → 87.06 → 83.17	23.000	26.000	3.000
Gemma-4-31B	85.00 → 66.00 → 62.00	94.21 → 92.43 → 90.53	19.000	23.000	4.000
Qwen3-30B-A3B	67.00 → 39.00 → 31.00	83.59 → 74.89 → 74.46	28.000	36.000	8.000
Devstral-2-123B	82.00 → 62.00 → 52.00	92.99 → 89.41 → 85.58	20.000	30.000	10.000
Llama-3.3-70B	81.00 → 62.00 → 54.00	92.26 → 89.97 → 88.05	19.000	27.000	8.000
GPT-OSS-120B	92.00 → 80.00 → 70.00	97.41 → 94.19 → 91.19	12.000	22.000	10.000
GPT-4.1 nano	16.00 → 7.00 → 4.00	36.53 → 29.74 → 27.27	9.000	12.000	3.000
Model-equal macro	73.11 → 52.00 → 46.33	85.29 → 80.49 → 77.98	21.111	26.778	5.667

For simulator steps, planner calls, and action-planning tokens, ratios of model-equal means are $(1.68, 1.57, 1.79)$ for B/A and $(1.10, 1.02, 1.02)$ for C/B (Table 23 in Appendix D.1). The ratio of steps to planner calls increases for every backbone, while aggregate token usage per call remains nearly unchanged. Thus, online revision is associated with a larger relative increase in simulator steps than in planner calls or action-planning tokens.

4.2 How Revisions Interact with Prior Execution

The effect of an online requirement depends on the execution state in which it arrives. Final-Upfront allows state manipulation and final placement to be planned jointly; Online-Revised may disclose additional requirements only after these arrangements have been made. We examine the distribution of online gaps by grouping tasks according to temporal, object-state, and adjacency requirements in their final specifications (Table 3). These subgroup comparisons are descriptive and do not isolate the effects of individual constraint types.

Temporal tasks are primarily challenged by the joint scheduling demands of the final specification. Their expansion gap is substantially larger, but their additional online gap is comparable to that of tasks without temporal constraints. Some original specifications impose cross-object stage dependencies, requiring multiple prerequisites to be completed before the next stage can proceed; added temporal requirements often form local chains among newly introduced objects without changing the order of existing work. Complex dependencies can remain difficult under upfront disclosure, while a new local ordering need not require completed work to be revisited. Greater difficulty under a temporal specification therefore need not imply greater sensitivity to online revision.

Adjacency tasks exhibit a different pattern. Tasks with and without adjacency requirements have similar expansion gaps, but their online SR gaps are 9.88 and 3.30 percentage points, respectively. The revision recipes add no adjacency propositions, yet some state requirements are imposed on objects already involved in adjacency relations. New operations must be considered jointly with retained locations, shared destinations, and spatial relations, so further processing of one object may involve preserving or restoring several conditions. The greater online execution increase in the adjacency group is consistent with the adaptation pressure associated with such coupling: the scope affected by a revision can extend beyond the requirements it directly adds.

Object-state tasks also exhibit larger online losses, with adaptation demands depending on disclosure timing and agent capabilities. When transport and state requirements for a new object arrive together, they can be incorporated into its initial handling plan; adding state requirements after an object has been allocated or placed may instead require that plan to be adjusted. If a new operation requires the human agent to take over work previously assigned to the Spot robot, an allocation mismatch may arise; if the human agent already handles the object, the revision may simply extend the existing work. Additional state requirements therefore do not automatically imply a need to renegotiate. Evaluating negotiation requires testing whether it repairs mismatches between existing allocations and new requirements, and whether its scope matches the interdependent work affected by a revision.

Table 3: **Completion gaps and online execution effort by final-task structure.** Each row reports model-equal means across nine backbones; n is the number of tasks per backbone in the subgroup. The expansion gap is $\mathrm{SR}_A - \mathrm{SR}_B$, and the online gap is $\mathrm{SR}_B - \mathrm{SR}_C$; the PC online gap is defined analogously. All gaps are in percentage points. Effort ratios divide subgroup model-equal means in C by those in B. The three features define separate, potentially overlapping splits. Gaps are computed from the subgroup summaries displayed in the appendix.

Task structure	n	SR (%) A→B→C	SR gap Expansion	SR gap Online	PC gap Online	Steps C/B	Tokens C/B
No temporal constraints	29	$75.86 \to 62.26 \to 56.23$	13.60	6.03	4.12	1.150	1.010
Temporal constraints	21	$69.31 \to 37.83 \to 32.67$	31.48	5.16	0.30	1.049	1.034
No object-state requirements	11	$76.26 \to 62.12 \to 59.34$	14.14	2.78	0.44	0.935	0.966
Object-state requirements	39	$72.22 \to 49.15 \to 42.66$	23.07	6.49	3.10	1.133	1.034
No spatial adjacency	32	$69.45 \to 48.79 \to 45.49$	20.66	3.30	2.44	1.049	0.971
Spatial adjacency	18	$79.63 \to 57.72 \to 47.84$	21.91	9.88	2.64	1.195	1.116

5 When Does Negotiation Help under Online Task Revision?

Online task revisions can alter existing allocations and dependencies among goals while adding new requirements. Under Online-Revised, we use explicit allocation dialogue as a coordination probe to examine how negotiation scope affects completion, which task structures benefit, and whether dynamic selection improves over fixed protocols. We compare NEVER, Full, Partial, Dynamic-skip, and Dynamic-max on the same 50 tasks using DeepSeek-v4-flash, Gemma-4-31B, GPT-OSS-20B, and Devstral-2-123B. All protocols receive the active task instruction; NEVER continues independent planning under updated requirements. The comparison therefore examines the incremental contribution of negotiation protocols when task updates are already available.

5.1 Broader Negotiation Does Not Guarantee More Effective Planning

Table 4: **Completion and execution-planning effort under Online-Revised.** Results average all reported runs within each backbone and then weight the four backbones equally. ΔSR is the difference from NEVER in percentage points (pp). Effort ratios divide model-equal means by the corresponding NEVER mean. Differences and ratios are computed before rounding. Calls are mean planning calls per agent; tokens sum both agents' action-planning input and output usage, excluding negotiation dialogue and selection overhead.

Protocol	SR (%)	PC (%)	ΔSR (pp)	Steps ratio	Planning calls ratio	Action tokens ratio
NEVER	57.50	87.16	0.00	1.000	1.000	1.000
Full	58.13	84.84	+0.63	0.975	1.045	1.124
Partial	59.25	85.05	+1.75	0.881	0.948	0.940
Dynamic-skip	62.50	87.56	+5.00	0.947	0.978	0.980
Dynamic-max	62.50	86.68	+5.00	0.938	0.965	0.987

Full and Partial share a proposal–review controller. Full permits global reconsideration of unfinished work, whereas Partial allows limited changes to existing allocations. Partial retains valid owners and remaining goal chains, permitting changes when new requirements invalidate an assignment or when additional capability-constrained work creates a substantial imbalance.

With equal weights across the four backbones, Partial reduces planning calls and action-planning tokens by 9.3% and 16.3% relative to Full, while increasing SR and PC by 1.13 and 0.21 percentage points, respectively (Table 4). During ongoing execution, allocations are linked to object possession, skills in progress, and remaining goal chains. Task revisions neither reset these states nor interrupt skills already in progress. Partial is designed to preserve valid execution commitments through local repair and reduce disruption to existing work assignments. Lower planning effort alongside higher mean completion is consistent with an interpretation based on plan stability.

All four negotiation configurations increase mean SR relative to NEVER, but PC increases only under Dynamic-skip, by 0.40 percentage points. For each backbone, we infer mean PC conditional on failure from SR and PC, then average these conditional means equally across backbones: the result is 70.1% for NEVER and 64.9–67.4% across the four negotiation configurations. Higher complete-success rates thus coexist with lower mean completion among the remaining failures, without a general improvement in mean partial completion. Both Dynamic variants achieve 62.50% SR, a gain of 5.00 percentage points over NEVER, and exceed both Full and Partial in SR and PC. Dynamic-max also reduces simulator steps, planning calls, and action-planning tokens by 6.2%, 3.5%, and 1.3%, respectively, relative to NEVER. In this successive-revision setting, dynamic selection achieves higher mean completion than fixed negotiation protocols while keeping downstream execution and planning effort below the no-negotiation baseline.

5.2 Negotiation Gains Vary with Capability Requirements and Goal Dependencies

For each task e, define structural labels $\ell(e) = (T_e, S_e, A_e) \in \{0,1\}^3$ indicating whether the final specification contains temporal constraints, object-state requirements, and spatial-adjacency constraints. Let $q \in \mathcal{Q} = \{\text{Full}, \text{Partial}, \text{Dynamic-skip}, \text{Dynamic-max}\}$ denote a negotiation configuration. A task stratum is indexed by $g = (z, b)$, where $z \in \{T, S, A\}$ and $b \in \{0,1\}$ indicates the

absence or presence of that feature; for example, $g = (A, 1)$ contains tasks with spatial adjacency. Let $\overline{\mathrm{SR}}_{q,g}$ denote SR under configuration q in stratum g, averaged equally across the four backbones. We define the within-stratum negotiation gain and the difference in gains between tasks with and without a feature as

$$\Delta_{q,g} = \overline{\mathrm{SR}}_{q,g} - \overline{\mathrm{SR}}_{\mathrm{Never},g}, \qquad \Gamma_{q,z} = \Delta_{q,(z,1)} - \Delta_{q,(z,0)}. \tag{2}$$

A positive $\Gamma_{q,z}$ indicates a larger negotiation gain on tasks containing feature z.

Spatial adjacency requires joint maintenance of coupled goals. All four negotiation configurations have higher mean SR than NEVER on tasks with spatial adjacency, with gains exceeding those in the non-adjacency stratum by 6.33–10.63 percentage points. Adjacency ties together the final placements of multiple objects. When a revision adds a state requirement to one of these objects, subsequent execution must accommodate both its processing and the retained spatial relation, potentially requiring retrieval, processing, and restoration of the layout. Allocation dialogue can align responsibility, operation order, and final placement across these related activities; its benefits are consistent with jointly maintaining coupled goals. Without adjacency, Full and Partial both have lower mean SR than NEVER, so fixed negotiation does not provide comparable gains. The adjacency stratum also has a slightly higher baseline SR, indicating that larger negotiation gains concentrate around goal coupling rather than simply tracking a lower baseline or greater headroom.

Object-state requirements make capability matching central to allocation. All four negotiation configurations yield positive mean SR gains on tasks with object-state requirements, with no SR decline on any backbone; all four $\Gamma_{q,S}$ values are positive. Operations such as cleaning and filling are available only to the human agent, so a new state requirement introduces a skill constraint into a transport goal that either agent could otherwise undertake. If the existing allocation cannot cover the added operation, the human agent must assume the relevant processing, while some of its transport work can be reassigned to the Spot robot. Negotiation can integrate capability matching with the coordination of related work, rather than merely balancing subtask counts. By contrast, Full yields no positive SR gain on any backbone in the no-state stratum. Capability-constrained work is therefore a comparatively consistent condition for negotiation gains in the evaluated tasks.

Temporal tasks require compatible allocations and execution order. Temporal constraints govern the order in which goals are first satisfied; even a valid assignment of responsibilities need not ensure that parallel execution respects these dependencies. In this stratum, Full reduces mean SR by 2.35 percentage points relative to NEVER, whereas Partial, Dynamic-skip, and Dynamic-max improve it by 2.10, 7.48, and 4.48 points, respectively. By preserving remaining goal chains, Partial allows limited allocation changes to remain anchored in the existing execution order. Global reconsideration must jointly account for task ownership and precedence across agents. Although temporal tasks have the lowest baseline SR, they do not benefit from the broadest negotiation scope.

Table 5: **Negotiation gains by task structure.** NEVER SR is the no-negotiation success rate within each stratum; the four protocol-difference columns give gains over the same-stratum NEVER baseline in percentage points. All values weight the four backbones equally. The three feature dimensions may overlap, and n is the number of tasks in each stratum.

Task structure	n	Never SR (%)	ΔFull	ΔPartial	ΔDynamic-skip	ΔDynamic-max
No temporal constraints	29	69.20	+2.78	+1.48	+3.20	+5.38
With temporal constraints	21	41.35	−2.35	+2.10	+7.48	+4.48
No object-state requirements	11	66.48	−7.93	−6.23	+0.55	+2.83
With object-state requirements	39	55.00	+3.03	+3.98	+6.23	+5.58
No spatial adjacency	32	56.65	−1.98	−1.58	+1.18	+2.73
With spatial adjacency	18	59.03	+5.25	+7.63	+11.80	+9.05

5.3 DYNAMIC SELECTION IMPROVES THE COMPLETION–EFFORT TRADE-OFF OVER FULL NEGOTIATION

Full permits reconsideration of all known unfinished work at every negotiation opportunity, whereas Dynamic selects whether to negotiate and how broadly to revise allocations based on the current

context. Using Full as the reference, we define the SR gain of Dynamic configuration d within task stratum g as

$$\mathcal{A}_{d,g} = \overline{\mathrm{SR}}_{d,g} - \overline{\mathrm{SR}}_{\mathrm{Full},g}, \qquad d \in \{\text{Dynamic-skip}, \text{Dynamic-max}\}. \tag{3}$$

Both Dynamic variants improve SR over Full on all four backbones, with an equal-weight mean gain of 4.38 percentage points. Dynamic-skip and Dynamic-max increase mean PC by 2.72 and 1.84 points and reduce action-planning tokens by 12.8% and 12.2%, respectively. Mean simulator steps and planning calls also decrease. Dynamic configurations therefore achieve a better completion–effort trade-off than Full: higher complete success and partial completion are achieved with less downstream planning effort.

Both Dynamic variants have higher mean SR than Full in all six task strata. Their gains are 8.48 and 10.75 percentage points in the no-state stratum and 9.83 and 6.83 points in the temporal stratum, respectively. They also further improve success in the state and adjacency strata, where Full already yields positive gains. The benefits thus cover two kinds of conditions: dynamic selection improves success rates on tasks where Full performs weakly while retaining and extending its gains on capability-constrained tasks and tasks with coupled goals. Compared with consistently allowing the broadest revision scope, context-dependent protocol selection yields more consistent completion gains across task conditions.

5.4 Experience-Based Selection and Disagreement Resolution

At episode start and task-revision opportunities, Dynamic retrieves same-role records from a separate experience bank using the current instruction, dialogue summary, local room and object information, and action history. Each agent recommends NEVER, Partial, or Full. Dynamic-skip executes the recommended protocol when the recommendations agree and otherwise selects NEVER; Dynamic-max selects the broader recommendation under the ordering $\mathrm{Never} < \mathrm{Partial} < \mathrm{Full}$. The two configurations share the retrieval mechanism and differ in how they combine local judgments into a team decision. Identical overall SR conceals different conditional outcomes: Dynamic-skip has larger gains in the temporal and adjacency strata, whereas Dynamic-max performs better in the non-temporal and no-state strata. Disagreement resolution thus changes how negotiation gains are distributed across task conditions.

6 Conclusion

In this work, we introduce a benchmark for embodied multi-agent collaboration under successive task revisions. Using revision recipes validated over complete sequences, the benchmark synchronously updates task instructions and executable evaluators during continuous physical execution while preserving valid evaluation histories across task versions, providing reproducible conditions for studying how new requirements interact with prior execution. Through upfront and online conditions with matched intended final requirements, we distinguish performance losses associated with final-task expansion from those associated with online adaptation. These losses are distributed differently across task structures: harder final tasks are not necessarily more sensitive to online revision. Negotiation gains are more concentrated in the object-state and spatial-adjacency task strata, echoing the distribution of additional online burden. The compatibility of new requirements with existing allocations, object possession, and goal dependencies offers a unifying perspective on these conditional effects. Allowing broader reallocation does not necessarily yield more effective coordination: in model-equal averages, Partial achieves higher completion than Full with fewer simulator steps, planning calls, and action-planning tokens. Both Dynamic configurations achieve higher success rates than Full on all four evaluated backbones, with lower mean action-planning token usage, further supporting the value of context-dependent selection. Coordination under task revision can thus be understood as context-dependent repair of execution commitments: identifying which existing arrangements remain valid and which activities require joint adjustment because of new capability requirements or goal dependencies, then determining whether to negotiate and how broadly to revise allocations. This study has limitations: our evaluation is restricted to two LLM-controlled agents with fixed capability profiles collaborating in simulated household environments. Whether these findings generalize to real human–robot interaction remains untested, and real users, larger teams, and physical robots may introduce additional challenges.

AI USE STATEMENT

In this work, we used generative AI tools for assistance with research questions and experimental design, literature comparison, technical and mathematical exposition, result interpretation, and internal review. Additionally, we used these tools to assist with drafting, polishing, and translating selected passages, as well as data reformatting, descriptive calculations, and the preparation of selected plotting code and illustrations. Language models also form part of the evaluated system, performing action planning and negotiation for the simulated human and robot agents. During benchmark construction, we used two language models to independently check the semantic consistency between cumulative task instructions and evaluation specifications. All retained task sequences underwent human review, including examination of the issues and disagreements identified by the models. We validated task feasibility through manual execution in the simulator under both successive revisions and upfront final requirements, as described in Section 3 and Appendix A.4. Synthetic examples used during figure development were excluded from the experimental results; the reported performance data come from simulator-based experiments. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This study is conducted in simulated household environments built on Habitat 3.0 and PARTNR. In the reported model evaluations, the term "human agent" refers to a simulated character controlled by a language model. The study does not involve human-subject experiments or participants' personal data. Manual execution is used to validate task feasibility during benchmark construction. Agent skill permissions and task revision rules are defined by the experimental configuration to construct controlled collaboration tasks with heterogeneous capabilities and evolving requirements. The empirical results characterize collaboration under the evaluated simulation environment, agent configurations, and task distribution.

REPRODUCIBILITY STATEMENT

Section 3 and Appendices A and B describe the data sources and partitioning, construction and validation of task revision sequences, progress-triggered scheduling, matched task conditions, history-aware evaluation, and evaluation metrics. Section 5 and Appendix E specify the negotiation protocols, execution mechanisms, and associated parameters; the latter also provides the complete negotiation and execution prompt templates. Appendix C documents experience collection, segment-return computation, role-conditioned retrieval, disagreement resolution, and the associated parameters. Sections 4 and 5, together with Appendix D, report experimental measurements, configuration-specific run counts, task-stratified results, and aggregation procedures. Backbone-specific experience banks and selector parameters are reported in Appendix C and Table 16.

REFERENCES

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. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a3cf318fbeec1126da21e9185ae9908c-Abstract-Conference.html.

Vardhan Dongre and Dilek Hakkani-Tür. Embodied multi-agent coordination by aligning world models through dialogue. In *Proceedings of the 27th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pp. 293–306, Atlanta, Georgia, USA, 2026. Association for Computational Linguistics. URL https://aclanthology.org/2026.sigdial-1.21/.

Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling*, pp. 212–221. AAAI Press, 2006. URL https://cdn.aaai.org/ICAPS/2006/ICAPS06-022.pdf.

HoBeom Jeon, Hyungmin Kim, Dohyung Kim, Minsu Jang, and Jaehong Kim. Communicative PARTNR: Natural language communication under partial observability in human–robot collaboration. In *ICCV Workshop on Human-Robot-Scene Interaction and Collaboration*, 2025. URL https://openreview.net/forum?id=r1f58OTBbY. Non-archival workshop paper.

Taehyeon Kim, Gyeongmin Kim, Eunsun Smith, and Byung-Cheol Min. HERON: Human-robot collaboration with efficient and resilient optimization for long-horizon planning. OpenReview preprint, 2026. URL https://openreview.net/forum?id=QobJeymX6Z.

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. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/b07091c16719ad3990e3d1ccee6641f1-Abstract-Conference.html.

Zhao Mandi, Shreeya Jain, and Shuran Song. RoCo: Dialectic multi-robot collaboration with large language models. In *2024 IEEE International Conference on Robotics and Automation*, pp. 286–299. IEEE, 2024. doi: 10.1109/ICRA57147.2024.10610855. URL https://ieeexplore.ieee.org/document/10610855.

Siddharth Nayak, Adelmo Morrison Orozco, Marina Ten Have, Vittal Thirumalai, Jackson Zhang, Darren Chen, Aditya Kapoor, Eric Robinson, Karthik Gopalakrishnan, James Harrison, Brian Ichter, Anuj Mahajan, and Hamsa Balakrishnan. Long-horizon planning for multi-agent robots in partially observable environments. In *Advances in Neural Information Processing Systems*, volume 37, pp. 67929–67967, 2024. doi: 10.52202/079017-2169. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/7d6e85e88495104442af94c98e899659-Abstract-Conference.html.

Yue Pei, Hongming Zhang, Jiarui Guan, Jusheng Zhang, Liang Lin, Haogang Zhu, and Ziliang Chen. Great minds think alike: Contextual tacit communication for decentralized LLM-agent cooperation. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL https://openreview.net/forum?id=TA2eE0lJmf.

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. URL https://openreview.net/forum?id=w_7JMpGZRh0.

Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander Clegg, Michal Hlavac, So Yeon Min, Vladimír Vondruš, Theophile Gervet, Vincent-Pierre Berges, John Turner, Oleksandr Maksymets, Zsolt Kira, Mrinal Kalakrishnan, Jitendra Malik, Devendra Chaplot, Unnat Jain, Dhruv Batra, Akshara Rai, and Roozbeh Mottaghi. Habitat 3.0: A co-habitat for humans, avatars, and robots. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/430894999584d0bd358611e2ecf00b15-Abstract-Conference.html.

Chak Lam Shek, Faizan M. Tariq, Sangjae Bae, David Isele, and Piyush Gupta. KGLAMP: Knowledge graph-guided language model for adaptive multi-robot planning and replanning, 2026. URL https://arxiv.org/abs/2602.04129.

Roman P. J. van der Krogt and Mathijs M. de Weerdt. Plan repair as an extension of planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, pp. 161–170. AAAI Press, 2005. URL https://cdn.aaai.org/ICAPS/2005/ICAPS05-017.pdf.

Zhimin Wang, Shaokang He, Duo Wu, Jinghe Wang, Linjia Kang, Jing Yu, and Zhi Wang. CoBel-World: Harnessing LLM reasoning to build a collaborative belief world for optimizing embodied multi-agent collaboration, 2025. URL https://arxiv.org/abs/2509.21981v1. Version 1.

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. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/54b8b4e0b4ba4aad112e84f32e3b5dbb-Abstract-Conference.html.

Xiaopan Zhang, Zejin Wang, Zhixu Li, Jianpeng Yao, and Jiachen Li. CommCP: Efficient multi-agent coordination via LLM-based communication with conformal prediction. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2026. URL https://comm-cp.github.io/.

Xinkui Zhao, Yifan Zhang, Sai Liu, Naibo Wang, Guanjie Cheng, Yueshen Xu, Chang Liu, Shuiguang Deng, and Jianwei Yin. DRAMA: Next-gen dynamic orchestration for resilient multi-agent ecosystems in flux, 2026. URL https://arxiv.org/abs/2508.04332v3. CVPR 2026; arXiv version 3.

Qinhong Zhou, Sunli Chen, Yisong Wang, Haozhe Xu, Weihua Du, Hongxin Zhang, Yilun Du, Joshua B. Tenenbaum, and Chuang Gan. HAZARD challenge: Embodied decision making in dynamically changing environments. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/d937cb3fe2851ed0ab9af5e38f885077-Abstract-Conference.html.

Qinhong Zhou, Chuang Gan, and Anoop Cherian. LLawCo: Learning laws of cooperation for modeling embodied multi-agent behavior. In *International Conference on Machine Learning (ICML)*, 2026. URL https://www.merl.com/publications/TR2026-081.

Henry Peng Zou, Chunyu Miao, Wei-Chieh Huang, Yankai Chen, Yue Zhou, Hanrong Zhang, Yaozu Wu, Liancheng Fang, Zhengyao Gu, Zhen Zhang, Kening Zheng, Fangxin Wang, Yi Nian, Shanghao Li, Wenzhe Fan, Langzhou He, Weizhi Zhang, Xue Liu, and Philip S. Yu. When users change their mind: Evaluating interruptible agents in long-horizon web navigation. *arXiv preprint arXiv:2604.00892*, 2026. doi: 10.48550/arXiv.2604.00892. URL https://arxiv.org/abs/2604.00892.

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. URL https://openaccess.thecvf.com/content/CVPR2025/html/Zu_Collaborative_Tree_Search_for_Enhancing_Embodied_Multi-Agent_Collaboration_CVPR_2025_paper.html.

A BENCHMARK CONSTRUCTION AND COVERAGE

The benchmark synchronously revises natural-language instructions and executable evaluation specifications during continuous execution to study how changing requirements affect task completion in heterogeneous embodied teams and when negotiation improves collaboration. Progress-triggered revisions follow recipes validated over complete sequences and preserve physical state and valid evaluation histories, so subsequent decisions depend on both new requirements and the consequences of prior execution. As shown in Figure 2, the experiments use two complementary comparisons: holding Never fixed, we compare Original-Static, Final-Upfront, and Online-Revised to distinguish performance changes associated with final-task expansion and online requirement introduction; holding Online-Revised fixed, we compare negotiation configurations to examine the effects of negotiation scope and experience-guided protocol selection. We further stratify tasks by temporal constraints, object-state requirements, and spatial-adjacency requirements in their final specifications to examine how online adaptation burden and negotiation gains vary across task structures and backbones.

A.1 PARTNR DATASET AND HETEROGENEOUS COLLABORATIVE TASK REPRESENTATION

Dataset background. PARTNR is a benchmark of embodied human–robot collaboration built on Habitat 3.0 and the Habitat Synthetic Scenes Dataset (HSSD) (Chang et al., 2025; Puig et al., 2024). It provides household tasks at the scale of 100,000 natural-language instructions, covering 60 houses and 5,819 distinct object models. Tasks require agents to rearrange objects, satisfy relational placement requirements, and manipulate object states in household environments. PARTNR distinguishes four task types: constraint-free tasks, spatial tasks, temporal tasks, and heterogeneous tasks involving differences in agent capabilities.

Each task pairs a natural-language instruction with an executable evaluation specification. The specification has three components: *propositions* describe the required conditions, *dependencies* determine when propositions are evaluated, and *constraints* specify how propositions must be satisfied jointly. We build task revisions on this structured representation so that instruction changes correspond to executable, verifiable changes in evaluation.

Heterogeneous skills of the human agent and Spot. Our experiments use a simulated human agent and a Spot robot. The human agent is also controlled by a language-model planner. Each agent selects skills using its own local observations and execution history, and their capability differences are defined by their available skill libraries. Table 6 lists the skills and permissions in the configurations used in our experiments.

Table 6: Skills available to the human agent and Spot in our experimental configurations. A dash denotes an unavailable skill.

Skill	Main effect and execution conditions	Human	Spot
`Explore`	Explore a specified room and observe its furniture and associated objects.	✓	✓
`Navigate`	Move near a specified room, piece of furniture, or object.	✓	✓
`Pick`	Grasp an object within reach; each agent can hold at most one object at a time.	✓	✓
`Place`	Place the held object on or inside specified furniture, optionally next to a reference object.	✓	✓
`Rearrange`	Combine picking, carrying, and placing an object at a destination.	✓	✓
`Open`, `Close`	Open or close articulated furniture, such as drawers, cabinets, and fridges.	✓	✓
`Wait`	Remain idle temporarily.	✓	✓
`Clean`	Clean an object with the corresponding affordance; some object categories require a nearby water source.	✓	–
`Fill`	Fill an object with the corresponding affordance; requires a nearby water source.	✓	–
`Pour`	Pour from a held, filled container into a fillable target container.	✓	–
`PowerOn`, `PowerOff`	Turn an object with the corresponding power-state affordance on or off.	✓	–

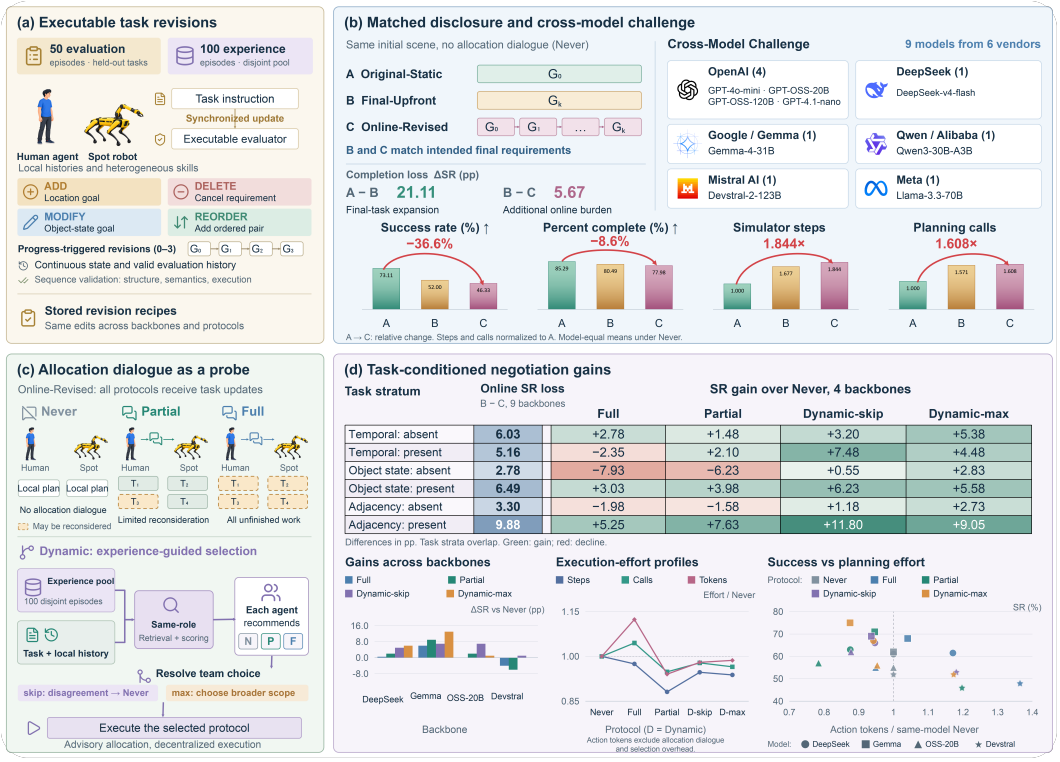


Figure 2: **Overview of the study design and main results.** (a) Construction and validation of executable task revisions. The evaluation and experience sets contain 50 and 100 tasks, respectively, and are disjoint. Revisions are triggered by execution progress and synchronously update cumulative instructions and evaluation specifications while preserving physical continuity and valid evaluation histories. (b) Comparison of three task conditions across nine backbones under Never. The difference between Original-Static and Final-Upfront characterizes performance changes associated with final-task expansion, while the difference between Final-Upfront and Online-Revised characterizes the additional online adaptation burden under matched intended final requirements. The SR gaps $A - B$ and $B - C$ are reported in percentage points. The declines shown above the SR and PC bars are relative changes from Original-Static to Online-Revised. Simulator steps and planning calls are normalized to Original-Static. (c) Negotiation protocols and dynamic selection. Never uses no allocation dialogue, Partial permits limited allocation adjustments, and Full permits reconsideration of all unfinished work. Dynamic selects a protocol using experience retrieved from the same agent role. Dynamic-skip selects Never when the agents' recommendations disagree, whereas Dynamic-max adopts the recommendation with the broader negotiation scope. All protocols receive task updates. (d) Negotiation gains and execution effort by task structure. The stratified results present online success-rate losses across nine backbones alongside negotiation gains across four backbones. The lower plots show gains for individual backbones, execution effort across five configurations, and the relationship between success rate and planning token usage. Success-rate differences in this panel are reported in percentage points, and groups defined by different task-structure dimensions may overlap. Execution effort is normalized to Never, and token usage in the scatter plot is normalized to Never for the same backbone. Throughout the figure, aggregate statistics weight backbones equally. Token counts cover action planning only and exclude allocation dialogue and protocol-selection overhead.

Both agents can explore, navigate, pick and place objects, and open or close furniture. The human agent additionally supports object-state operations. These state skills require the target to be within reach and to have the corresponding affordance; whether a skill call is needed also depends on the object's current state. This capability configuration makes collaboration involve the allocation and coordination of object transport, state manipulation, and subsequent placement.

Predicates and atomic propositions. PARTNR predicates describe object locations, relations between objects, and object states. Instantiating a predicate with entities and arguments yields an atomic proposition that the simulator can evaluate. Table 7 summarizes the predicate vocabulary, where o denotes an object instance, f denotes furniture or a placement target, and r denotes a room.

Table 7: PARTNR predicate vocabulary. Predicates describe target conditions independently of the skills used to achieve them.

Type	Predicate	Meaning
Location	`is_on_top`(o, f)	Object o is on a supporting surface of f.
Location	`is_inside`(o, f)	Object o is inside f.
Location	`is_in_room`(o, r)	Object o is in room r.
Location	`is_on_floor`(o)	Object o is on the floor.
Spatial	`is_next_to`(o_1, o_2)	The objects satisfy the simulator's adjacency relation.
Spatial	`is_clustered`$(\ldots)$	One connected component of the adjacency graph satisfies the required count for every object group.
State	`is_clean`(o)	Object o supports the cleanliness state and is currently clean.
State	`is_dirty`(o)	Object o supports the cleanliness state and is currently dirty.
State	`is_filled`(o)	Object o supports the fill state and is currently filled.
State	`is_empty`(o)	Object o supports the fill state and is currently empty.
State	`is_powered_on`(o)	Object o supports the power state and is currently on.
State	`is_powered_off`(o)	Object o supports the power state and is currently off.

The simulator evaluates these predicates using geometric relations, semantic regions, and object states. All `is_next_to` propositions in our selected tasks use a horizontal distance threshold of $1.0\,\mathrm{m}$. Table 7 lists the full PARTNR predicate vocabulary; coverage in our task sets is reported in Appendix A.5.

Proposition arguments can contain multiple candidate instances, expressing the choices allowed by a natural-language instruction. For example,

```
is_on_top([cup_1, cup_2], [table_1])
```

holds under the default count setting when either candidate cup is on the specified table. The argument `number`$= k$ requires at least k distinct candidate objects to satisfy the condition; setting `number`$= 2$ in this example requires both cups to be on the table. For a proposition with multiple candidate destinations, `is_same_receptacle=True` can additionally require the objects to use the same destination. These arguments express count and destination-consistency requirements *within one proposition*; argument constraints express consistency across propositions.

Proposition constraints. A single proposition is often insufficient to express a complete instruction. For example, "clean the cup before placing it on the table" includes an ordering requirement, while "put the two books on the same table" requires consistency across placement propositions. PARTNR expresses these relations with the four constraint types in Table 8.

Argument constraints compare the instances that actually satisfy the propositions. Two placement propositions may share the same list of candidate tables, but a corresponding `SameArgConstraint` is satisfied only when their selected tables coincide. Terminal constraints check the state at termination, allowing agents to temporarily change a satisfied placement relation while transporting or rearranging objects.

Proposition dependencies. Dependencies determine the conditions under which the evaluator checks a target proposition. If a dependency condition is unmet, the target does not participate in that evaluation. Each dependency specifies target propositions, prerequisite propositions, a rela-

Table 8: PARTNR constraint types. Argument constraints compare the actual satisfying instances of the named arguments.

Constraint	Semantics and example
`TemporalConstraint`	A directed acyclic graph over proposition indices. An edge $[i, j]$ requires P_i to be first recorded as satisfied strictly before P_j; an empty edge set imposes no additional order. Example: clean the cup before placing it on the table.
`TerminalSatisfactionConstraint`	The listed propositions must hold at episode termination. They may temporarily become unsatisfied and be restored before termination. Example: the cup must end on the specified table.
`SameArgConstraint`	The actual satisfying instances of named arguments across propositions must coincide. Example: two books must be placed on the same table.
`DifferentArgConstraint`	The actual satisfying instances of named arguments across propositions must be pairwise distinct. Example: different candles must be placed on different tables.

tion type, and an `all` or `any` mode for combining multiple prerequisites. Table 9 lists the relation types.

Table 9: Dependency relations that determine when target propositions are evaluated.

Relation	Condition for evaluating the target propositions
`while_satisfied`	The prerequisites hold in the most recent evaluation state.
`after_satisfied`	The prerequisites have held at some point in the evaluation history.
`after_unsatisfied`	The prerequisites have held and subsequently transitioned from satisfied to unsatisfied.
`before_satisfied`	The prerequisites have never held in the preceding evaluation history.

For example, "put the cup and plate on the dining table next to each other" can be decomposed into two placement propositions and one adjacency proposition. Giving the adjacency proposition a `while_satisfied` dependency on both placement propositions allows adjacency to be evaluated only once both objects are on the specified table, preventing incidental adjacency elsewhere from counting as completion. The evaluator combines proposition-satisfaction histories with the relevant constraints to compute percent complete (PC). Completion therefore depends on whether target conditions have been achieved and whether their order, instance choices, and terminal states meet the specification.

Task selection. We select 100 experience episodes from PARTNR's human-annotated subset of 2,000 training episodes and 50 evaluation episodes from its validation split. Selection is stratified by source task type and proposition count, while also covering scene diversity and features such as targets in multiple rooms, cleaning, filling, power states, containment, and argument constraints. Sampling labels define strata; spatial, temporal, and object-state requirements in the finalized tasks can overlap. The experience set supplies trajectories for experience construction, and reported evaluation results use the evaluation set. The two sets share no $(\text{scene}, \text{episode})$ pair and no exactly identical initial instruction. The experience set covers 37 scenes and the evaluation set 12, with one shared scene. The resulting partition is episode-disjoint.

A.2 Task revision operators and dynamic evaluator updates

Revision targets and task versions. We construct revision sequences using PARTNR's structured task representation. Following the task definition $G = (I, \Phi, \mathcal{C})$ in the main text, I is the natural-language instruction, $\Phi = (\phi_0, \ldots, \phi_{n-1})$ is an ordered proposition list, and $\mathcal{C}$ comprises proposition dependencies $\mathcal{D}$ and constraints $\mathcal{K}$. Each revision updates the instruction and its executable evaluation specification together, maintaining consistency across target conditions, proposition relations, and completion assessment. Revision sequences are generated and stored before task execution. Each event records its trigger threshold, realized atomic operations, proposition additions

and deletions, constraint changes, and updated cumulative instruction. At runtime, events update the task in their stored order while preserving the continuity of the physical scene and execution history.

Four atomic revision operators. We use four atomic operators, ADD, DELETE, MODIFY, and REORDER, to construct candidate revisions. Table 10 summarizes their semantics and default generation conditions.

Table 10: Atomic revision operators and their default generation conditions. Finalized specifications can preserve location semantics adjusted during validation.

Operation	Change to the task specification	Default generation conditions
ADD	Introduce a location requirement for an object, appending $\texttt{is_on_top}(o, f)$ by default.	Prefer previously unreferenced objects; select the destination from task-related furniture.
DELETE	Remove a selected proposition and update its dependencies, constraints, and evaluation history.	Retain at least one proposition, and avoid removing a placement requirement on which a remaining adjacency requirement relies.
MODIFY	Append a clean, filled, or powered-on requirement for an existing task object while retaining its other requirements.	The new state must be supported by the object's category and must not already be required by the current specification.
REORDER	Introduce two objects, append their placement propositions at a shared destination, and add an ordering edge between them.	Prefer two objects not previously referenced by any proposition in the episode. Select their shared destination from the original task's placement furniture.

MODIFY strengthens the requirements for an existing object by adding a state condition. REORDER extends temporal requirements through new goals with an explicit order. DELETE cancels a task requirement without removing the corresponding physical object from the scene.

Let $\|$ denote list concatenation, $\mathcal{E}$ the temporal edge set, and $n = |\Phi|$ the proposition count immediately before the current atomic operation. The default forms of the three additive operators are

$$\begin{aligned} \text{ADD}_{o,f}: &\quad \Phi' = \Phi \| \langle \texttt{is_on_top}(o, f) \rangle, \\ \text{MODIFY}_{o,\sigma}: &\quad \Phi' = \Phi \| \langle \sigma(o) \rangle, \quad \sigma \in A(c_o) \setminus S_\Phi(o), \\ \text{REORDER}_{a,b,f}: &\quad \Phi' = \Phi \| \langle \texttt{is_on_top}(a, f), \texttt{is_on_top}(b, f) \rangle, \\ &\quad \mathcal{E}' = \mathcal{E} \cup \{(n, n+1)\}. \end{aligned} \tag{4}$$

Here $A(c_o)$ is the set of state predicates supported by object category c_o, a subset of `is_clean`, `is_filled`, and `is_powered_on`; $S_\Phi(o)$ contains the corresponding state requirements already imposed on object o. MODIFY itself introduces no temporal edge between state manipulation and placement; their execution order is determined by the explicit temporal constraints in the specification.

Equation 4 describes the default generation of candidate revisions. During validation, added goals can adopt containment, room-level, or candidate-furniture-group requirements to match the instruction semantics. For example, the finalized ADD in experience episode 1727 uses `is_inside`, while that in evaluation episode 940 uses `is_in_room`. The predicate used by ADD therefore depends on the intended location semantics, and runtime execution uses the finalized stored propositions and constraints.

Consistency of dependencies and constraints. Proposition additions and deletions require corresponding updates to indices and constraint relations. After deleting proposition d, the remaining propositions retain their relative order and are re-indexed by

$$\pi_d(i) = i - \mathbf{1}[i > d], \qquad i \neq d. \tag{5}$$

References in dependencies and constraints are mapped to the new indices. Temporal edges incident to the deleted proposition are removed, argument constraints drop the corresponding argument names, and dependencies with empty target or prerequisite lists are removed. All added propositions enter terminal-satisfaction constraints, requiring the new goals to hold at episode termination. When

an added instruction refers to "the same place" and its reference proposition admits multiple candidate pieces of furniture, the generator adds a `SameArgConstraint` to bind the new placement to the destination that actually satisfies the reference proposition.

A deletion can also change the task's final stage. If a cancelled proposition was a terminal requirement, the generator searches backward for its nearest surviving predecessors in the graph formed by temporal relations and proposition dependencies. Only predecessors with no surviving successor after deletion are added to the terminal requirements. This mechanism maintains an appropriate terminal condition for the remaining task after its original final stage is cancelled. These updates jointly maintain consistency among propositions, dependencies, and constraints.

Compound events and cumulative instructions. A revision event can contain multiple atomic operations. The generator first samples the requested operation count K:

$$\Pr(K = 1) = \Pr(K = 2) = \Pr(K = 3) = \frac{1}{3}. \tag{6}$$

It then samples K operator types independently and uniformly with replacement, allowing repeated types within an event. Candidate operations are constructed in the order

$$\textsc{Delete} \rightarrow \textsc{Add} \rightarrow \textsc{Modify} \rightarrow \textsc{Reorder},$$

and each operation is checked against the specification resulting from the preceding operations.

Infeasible operations are discarded and feasible operations are retained, so the realized count can be smaller than K. If all operations in an attempt fail, the generator resamples their types with K fixed, allowing at most eight attempts in total. A candidate event is discarded if no valid operation is obtained. Stored deletion indices refer to the proposition list at the start of an event. Runtime application first deletes propositions in descending index order and repairs references, then appends propositions and constraint changes in their stored order, preserving index consistency within the compound event. The cumulative instruction after each event includes the original task, preceding revisions, and newly added clauses, with cancellations explicitly identifying requirements that no longer apply. The active task is determined by these cumulative revisions. Validation adjusts wording and entity references where necessary to maintain agreement between the cumulative instruction and the reconstructed active evaluation specification.

Dynamic evaluation and history updates. At revision injection, object locations, object states, and agent possession remain unchanged while the evaluator is updated for the active task. The evaluation history records each proposition's satisfaction status and its satisfying entity instances; the latter support consistency checks on arguments across propositions.

The evaluator applies a revision in six steps:

1. **Remove cancelled requirements.** Delete propositions, their history entries, and their first-satisfaction indices in descending proposition-index order.
2. **Update references.** Remap proposition indices in dependencies and constraints, maintain the corresponding argument-name lists, remove invalid references, and rebuild affected temporal graphs.
3. **Initialize added requirements.** Append propositions in stored order, initialize their first-satisfaction indices as unset, and extend earlier history records with unsatisfied results and empty argument evidence.
4. **Apply constraint changes.** Add the new constraints and temporal edges, and update the evaluator's proposition count.
5. **Evaluate the current state.** Determine which propositions are eligible under their dependencies, evaluate them in the current scene, append the results, and update first-satisfaction indices.
6. **Update evaluation metrics.** Recompute completion and success from the updated satisfaction history and constraint validity.

The historical evidence and first-satisfaction indices of retained propositions are preserved after reindexing. Recording for a new proposition begins at its introduction, although the physical state

established before the revision can satisfy it. For example, a newly introduced cleanliness requirement can hold at its first evaluation if the object is already clean, but its first-satisfaction index is never backdated to before its introduction.

Let $n(G)$ be the number of propositions in the active evaluator, τ_i^{sat} the index in evaluation history at which ϕ_i is first recorded as satisfied (-1 if unset), and $v_i \in \{0, 1\}$ its joint validity under the relevant constraints and current history. For $n(G) > 0$,

$$\mathrm{PC}_G = \frac{1}{n(G)} \sum_{i=0}^{n(G)-1} \mathbf{1}[\tau_i^{\mathrm{sat}} \geq 0]\, v_i, \qquad \mathrm{Success}_G = \mathbf{1}[\mathrm{PC}_G = 1]. \tag{7}$$

The implementation assigns zero completion to an empty proposition list. Temporal constraints assess ordering from the satisfaction history. Terminal constraints check the latest evaluation state during online scoring and the episode's final state at termination, allowing a proposition to become temporarily unsatisfied and later be restored. A revision can change both the active proposition count and constraint validity, causing a completion jump without a physical state change. Completion is therefore recomputed after every revision before testing the next event's trigger condition, as detailed in Appendix A.3. Automatic successful termination requires all scheduled revisions to have been applied and the active task to satisfy its success condition. Episodes terminated by a budget limit are evaluated under the task version active at termination.

Complete revision example. The original instruction of evaluation episode 615 is: "First, move the basketball and then the carrying case from the living room table to the bedroom chest of drawers. Place them next to each other on the chest of drawers." Let f denote that chest of drawers. The initial specification contains placement proposition ϕ_0 for the basketball, placement proposition ϕ_1 for the carrying case, and adjacency proposition ϕ_2. The initial temporal edges are $\{(0, 1), (0, 2)\}$, the adjacency proposition has a `while_satisfied` dependency on the carrying-case placement, and all three propositions are terminal requirements. Table 11 traces the subsequent revisions.

Table 11: Complete revision sequence of evaluation episode 615. Proposition counts describe the stored task specifications at successive versions.

Version	Revision content	Evaluation-specification change	$\lvert\Phi\rvert$
Initial	Place the basketball and carrying case on the chest of drawers with the required order and adjacency.	Two placement propositions, one adjacency proposition, and the original constraints.	3
Revision 1	MODIFY: require the carrying case to be clean.	Append $\phi_3 =$ `is_clean`(case) and make it terminal.	4
Revision 2	ADD: place `box_5` on the same chest of drawers.	Append placement proposition ϕ_4 and make it terminal.	5
Revision 3	REORDER: place the bottle on the chest of drawers before `box_6`.	Append placement propositions ϕ_5, ϕ_6, temporal edge $(5, 6)$, and their terminal requirements.	7

This sequence progressively adds placement goals, an object-state requirement, and a temporal constraint. The first compound revision of evaluation episode 90 illustrates the combination of DELETE with another operator: it cancels one placement requirement and adds one cleanliness requirement. The proposition count remains twelve, while the active goals and their terminal constraints change. Revision events, atomic operations, and evaluator propositions are distinct counting units. One event can contain several atomic operations, one REORDER introduces two propositions, and a state requirement for a group of objects can be represented by multiple proposition records. Appendix A.5 reports counts and changes for these units using the finalized events and evaluation specifications.

A.3 PROGRESS-DEPENDENT REVISION SAMPLING AND TRIGGERING

Design rationale. Execution speed varies across models and collaboration strategies, so a revision scheduled at a fixed simulator step can correspond to different stages of task completion. We schedule revisions using task completion to associate their triggering conditions with current task progress. Completion is computed by the evaluator in Appendix A.2, incorporating proposition-satisfaction

histories and the relevant constraints. Specifically, revision thresholds are sampled offline on the normalized completion interval, and events are triggered at runtime according to completion under the active task version. The proposal intensity increases with completion, producing denser candidate revisions at higher completion levels to assess adaptation in the presence of existing task progress and allocations.

Sampling intensity in completion space. Let $\rho \in [0, 1]$ denote the normalized completion coordinate. Candidate thresholds are generated by a non-homogeneous Poisson process (NHPP) with intensity

$$\lambda(\rho) = \lambda_0(1 + \alpha\rho), \qquad \lambda_0 = 0.8, \quad \alpha = 3. \tag{8}$$

The corresponding cumulative intensity is

$$\Lambda(\rho) = \int_0^\rho \lambda(u)\,\mathrm{d}u = \lambda_0 \left(\rho + \frac{\alpha}{2}\rho^2\right) = 0.8\rho + 1.2\rho^2. \tag{9}$$

Here λ_0 controls the overall intensity scale and α controls its growth with completion. The chosen parameters satisfy

$$\frac{\lambda(1)}{\lambda(0)} = 4, \qquad \Lambda(1) = 2. \tag{10}$$

Thus, before imposing a count cap, the intensity per unit completion at the end of the interval is four times that at its start, and the expected candidate count over the interval is two. This intensity is defined on the completion coordinate and measures expected events per unit completion. The count cap, feasibility filtering, and validation adjustments can all alter the finalized threshold distribution; the fourfold ratio describes only the intensity of the original proposal process.

Threshold sampling and the count cap. We implement this process by thinning with envelope intensity

$$\bar{\lambda} = 5 \geq \max_{\rho \in [0,1]} \lambda(\rho) = 3.2. \tag{11}$$

Starting at candidate coordinate $r = 0$, the sampler performs the following steps:

1. Draw $U_1 \sim \mathrm{Uniform}(0, 1)$, generate the candidate increment
$$\Delta r = -\frac{\ln U_1}{\bar{\lambda}},$$
and update $r \leftarrow r + \Delta r$.
2. Stop if $r \geq 1$.
3. Independently draw $U_2 \sim \mathrm{Uniform}(0, 1)$. Accept the candidate if
$$U_2 < \frac{\lambda(r)}{\bar{\lambda}},$$
round it to four decimal places, and store it as a revision threshold.
4. Repeat until the end of the interval is reached or three thresholds have been accepted.

Accepted thresholds are stored in sampling order, yielding a nondecreasing sequence after rounding. The count cap retains the earliest three accepted points and therefore affects the proportion of later thresholds retained. If no candidate is accepted, the sequence remains empty and no additional revision is inserted. Table 12 lists the sampling parameters.

Seeds are determined by the input order within each set, so identical positions use the same threshold seed. Revision content depends on the entities, propositions, and constraints of each task. Shared candidate thresholds do not imply identical realized injection times across tasks.

Theoretical candidate-count distribution. For the target process without a count cap, the number of candidates on $[0, 1]$ satisfies

$$N \sim \mathrm{Poisson}(\Lambda(1)) = \mathrm{Poisson}(2). \tag{12}$$

Table 12: Parameters for threshold sampling and revision-content generation. The input position b is zero-based.

Parameter	Setting
Base intensity λ_0	0.8
Progress coefficient α	3
Envelope intensity $\bar{\lambda}$	5
Candidate-threshold cap per episode	3
Stored threshold precision	Four decimal places
Threshold seed for input episode b	$42 + 1000b$
Revision-content seed	Corresponding threshold seed $+1$

With the three-threshold cap, the retained candidate count is $L_{\text{cand}} = \min(N, 3)$, giving

$$\begin{aligned} \Pr(L_{\text{cand}} = k) &= e^{-2}\frac{2^k}{k!}, \qquad k = 0, 1, 2, \\ \Pr(L_{\text{cand}} = 3) &= 1 - 5e^{-2}. \end{aligned} \tag{13}$$

The corresponding probabilities are approximately 13.5%, 27.1%, 27.1%, and 32.3%, and the expected count is

$$\mathbb{E}[L_{\text{cand}}] = 3 - 9e^{-2} \approx 1.78. \tag{14}$$

This distribution explains why the sampler permits both zero revisions and multiple revisions. These results describe candidates before feasibility filtering and human review. Finalized event counts and counts actually triggered in a rollout are additionally affected by task construction and execution trajectories, respectively.

Action boundaries and runtime triggering. Let the finalized revision sequence be

$$\Xi = \big((\rho_1, \xi_1), \ldots, (\rho_L, \xi_L)\big), \qquad L \le 3, \tag{15}$$

where ρ_j is the trigger threshold of event j and ξ_j is its task revision. Runtime checks consider the next pending event in stored order. For the language-model planners used in our experiments, a revision condition can be checked whenever at least one agent has no high-level skill in progress; the initial episode state is also an eligible boundary. The agents execute skills asynchronously, so their action-completion times need not coincide. Checking when either agent reaches an action boundary avoids prolonged revision delays caused by waiting for both agents to be idle simultaneously.

Let $\mathcal{B}$ denote these boundaries, with the initial boundary indexed by $t = 0$. The realized injection time of event j is

$$d_j = \inf\left\{t \in \mathcal{B} :\ t \ge d_{j-1}, \quad \mathrm{PC}_{G^{(j-1)}}(s_t, \eta_t) \ge \rho_j\right\}, \qquad d_0 = 0, \tag{16}$$

where $G^{(j-1)}$ is the task version after the first $j-1$ events and η_t is its evaluation history. If the preceding revision has already been applied at the same boundary, completion is evaluated after that revision and the resulting metric update. If no eligible boundary satisfies the condition before episode termination, we set $d_j = +\infty$ and the event is not triggered. The threshold is a lower bound on completion at injection. A single skill can satisfy several propositions, so realized completion can exceed the threshold. Propositions already satisfied in the initial state can also trigger the first event at the initial boundary. The decision always uses current completion at the boundary and does not require an earlier action to have crossed the threshold from below.

Re-evaluation after each event and completion changes. Completion refers to the active task version and is not a monotonically increasing scheduling clock. After every revision, the system updates proposition results, constraint validity, completion, and success before comparing the next threshold. For example, suppose that q of n propositions contribute to completion before a revision. If the event only adds m unsatisfied propositions and leaves the contribution of all existing propositions unchanged, then

$$\mathrm{PC}^- = \frac{q}{n}, \qquad \mathrm{PC}^+ = \frac{q}{n+m}, \tag{17}$$

and hence

$$\mathrm{PC}^{+} - \mathrm{PC}^{-} = -\mathrm{PC}^{-}\frac{m}{n+m}. \tag{18}$$

This change is caused by the revised specification and requires no physical state change. For compound events that also delete requirements or change constraints, the completion change is determined by the fully updated evaluator. As an illustration, suppose four of six propositions contribute to completion, giving $\mathrm{PC}^{-} = 4/6$. A threshold of 0.60 is reached, and the triggered event adds two unsatisfied propositions, reducing completion to $4/8 = 0.50$. If the next threshold is 0.65, that event remains pending and execution continues under the active task version.

Adjacent thresholds can be very close. Reusing pre-revision completion across successive checks would ignore the effects of specification changes and could trigger later events prematurely. The system therefore recomputes PC after each event before deciding whether to apply the next one. Multiple events can take effect at the same eligible boundary, so $d_j = d_{j-1}$ is permitted. If updated completion still meets the next threshold, the next event is applied; otherwise, the current check ends. Events already applied are not revoked if completion subsequently decreases.

Execution continuity and negotiation opportunities. Revision injection preserves object locations, object states, and agent possession, and does not interrupt the other agent's ongoing skill. That agent completes its current skill and uses the updated instruction at its next planning step. Atomic operations, revision events, and injection batches are distinct levels: an event can contain multiple atomic operations, and an action boundary can admit multiple events. If execution must continue after a batch, the system delivers the final cumulative instruction once and provides one post-revision negotiation decision opportunity. Whether dialogue occurs depends on the collaboration strategy. Once all scheduled revisions are active and the current task succeeds, the episode terminates successfully. Thresholds and future revision content are maintained internally by the environment. Agents receive the currently active instruction without advance access to subsequent revisions or their thresholds. Thus, an offline fixed revision sequence still requires agents to adapt their behavior to new instructions at runtime.

Finalized recipes and reproducibility. Feasibility filtering and validation adjustments transform candidate thresholds into the finalized stored event sequence. Experiments load this sequence directly and replay its fixed thresholds and revision content. Input order, random seeds, and generator parameters describe candidate construction; the finalized task recipes determine the evaluation specifications used in the experiments. Different models or collaboration strategies use the same stored sequence for a given task, while realized injection times and triggered-event counts can vary with their execution trajectories. Events whose trigger conditions are unmet are not forcibly applied.

A.4 Validation of complete task sequences

We validate each episode's complete task sequence, including its initial specification, the intermediate specifications after successive revisions, and its final specification. Validation comprises automatic structural checks, independent semantic review by two language models with human review, and manual execution in the simulator. These stages examine the structural consistency of the task representation, the semantic agreement between natural-language requirements and evaluator specifications, and the executability of the revised tasks, respectively. All 100 experience episodes and 50 evaluation episodes retained in the benchmark passed these checks.

Structural checks. Starting from each episode's initial specification, we apply its revision events in their stored order and check the complete specification after each update. The 150 episodes contain 272 stored revision events, so the structural checks cover 422 specification versions: 150 initial versions and 272 versions following an event.

The checks verify that propositions use supported predicates; proposition indices referenced by constraints and dependencies are valid; the argument-name lists of `SameArgConstraint` and `DifferentArgConstraint` align with their proposition-index lists and refer to arguments present in the corresponding propositions; temporal graphs are acyclic and their recorded proposition counts match the active specifications; and dependencies use supported relation types and contain neither empty references nor cycles. In addition, the proposition count after each update

must equal the preceding count minus the number of deletions plus the number of additions. These checks operate directly on the finalized stored artifacts to assess structural consistency across accumulated revisions.

Independent semantic review and human review. Claude Opus 5.0 and GPT 5.6 sol independently review all task sequences, and human review addresses the identified issues and disagreements between the models. The review compares the cumulative natural-language instruction with the complete evaluator specification of the corresponding version. Additions, cancellations, and changes are interpreted together to identify the currently active requirements, which are then checked against the propositions, dependencies, and constraints.

The review covers five aspects. First, the active requirements in the instruction must agree with the evaluator specification. Second, references to objects, furniture, and rooms must be unambiguous, with consistent interpretations of individual instances and candidate sets. Third, state requirements must be supported by the objects' interaction affordances and achievable by an agent with the corresponding skills. Fourth, ordering, shared or distinct targets, and requirements that must hold at termination must agree with their respective constraints. Fifth, successive specifications must remain consistent: requirements and relations unaffected by a revision are retained, while relations affected by a cancellation or change are updated accordingly, avoiding mutually exclusive terminal requirements or invalid dependency references. Table 13 summarizes representative consistency issues and how they are addressed.

Table 13: Representative consistency issues considered during validation and their treatment.

Issue	Check and treatment
Inconsistent entity references or target scope	Use distinguishable instance names and check that the instruction and specification agree on the intended furniture instance, candidate furniture set, or room.
Unsupported or unintended repeated state requirements	Check object affordances and existing requirements to exclude infeasible or unintended duplicate state propositions.
Inconsistent relations after cancellation	Check surviving adjacency requirements and dependencies; after deleting a terminal stage, inspect and supplement terminal constraints as required by the updated task structure.
Unconstrained "same place" references	When a destination admits multiple candidate instances, use `SameArgConstraint` as required by the instruction to bind the target instances that satisfy the related propositions.
Misaligned references or history after deletion	Update proposition indices, constraint arguments, dependencies, and evaluator histories together so that surviving propositions remain aligned with their records.

Execution through successive revisions. In the simulator, a human operator controls the human agent and Spot to execute the task from the episode's original initial state. Stored revision events are applied in sequence according to the progress checks and triggering rules in Appendix A.3. Execution continues under the revised requirements while preserving the environment state across revisions, until all stored events have been applied and the evaluator reports success on the final specification. Revision timing is determined by the progress conditions; it does not require all propositions of the active version to be completed first. This procedure checks whether revisions can be triggered during execution and whether the revised requirements can be completed from the environment state produced by earlier actions. It examines the reachability of objects and destinations, the operability of required containers, the availability of environmental resources needed for cleaning or filling, and the compatibility of actions with agent skills.

Execution of the complete final task. We additionally apply all stored revisions in order before execution to obtain the final cumulative instruction and complete evaluator specification. This final task is then executed from the episode's original initial state until the evaluator reports success. The procedure checks whether the final propositions, dependencies, and constraints can be jointly satisfied within a single execution. Execution through successive revisions validates the continuation and completion of tasks as their requirements change, while execution of the final task validates the

joint satisfiability of all retained requirements from the original scene. Together with the structural checks and semantic review, these procedures constitute the validation of complete task sequences.

A.5 Task coverage and specification complexity before and after revisions

We characterize the 100 retained experience episodes and 50 evaluation episodes by comparing their initial and fully revised task specifications. "Initial" denotes the task version before any revision, and "final" denotes the version obtained by applying all stored events in order. All statistics are recomputed from the task files and revision records. They describe the constructed tasks, independently of how many events are triggered in a particular execution.

Specification size and combinations of requirements. We report coverage along four dimensions: proposition composition, combinations of requirements, entity categories, and revision sequences. Proposition counts refer to serialized proposition records. Placement propositions comprise `is_on_top`, `is_inside`, `is_in_room`, and `is_on_floor`; adjacency propositions use `is_next_to`; and state propositions describe cleanliness, filling, and power states.

At the episode level, we examine three additional requirement features: the presence of at least one adjacency proposition, at least one state proposition, and at least one directed edge in the temporal constraint graph. These features can co-occur. An episode with zero features has none of these three additional requirements but can still contain basic placement goals. These indicators describe specification content and are distinct from the sampling categories in Appendix A.1. Table 14 reports the resulting coverage and revision statistics.

Table 14: Coverage and revision statistics for the initial and fully revised tasks. The experience and evaluation sets contain 100 and 50 episodes, respectively. Requirement features may overlap. The revision-sequence rows summarize the stored revision sequences; dashes indicate that an initial-specification entry is not applicable. Event counts are not counts of events triggered in a particular rollout. Entity categories follow the counting convention described below.

	Experience ($n = 100$)		Evaluation ($n = 50$)	
Quantity	Initial	Final	Initial	Final
Proposition size and composition				
Total propositions	437	718	226	358
Mean per episode (range)	4.37 (1–12)	7.18 (1–14)	4.52 (1–13)	7.16 (3–15)
Placement/adjacency/state	307/57/73	455/53/210	163/28/35	235/28/95
State share (%)	16.7	29.2	15.5	26.5
Requirement coverage: episodes				
State	35	84	13	39
Temporal order	23	55	12	21
Adjacency	36	33	18	18
State and temporal order	11	48	1	16
None of the three features	26	4	15	3
With 0/1/2/3 features	26/54/20/0	4/37/42/17	15/27/8/0	3/25/13/9
Task-relevant category coverage				
Objects/furniture/rooms	70/12/12	79/13/13	50/11/10	63/12/10
Complete revision sequences				
Candidate thresholds/retained events	–	184/181	–	91/91
Episodes with 0/1/2/3 events	–	13/21/38/28	–	4/16/15/15
Atomic operations	–	281	–	122
ADD/DELETE/MODIFY/REORDER	–	72/23/143/43	–	46/2/60/14

The total number of propositions increases by 64.3% in the experience set and 58.4% in the evaluation set, with the mean per episode rising from 4.37 to 7.18 and from 4.52 to 7.16, respectively. New requirements concern both placements and object states. The share of state propositions increases from 16.7% to 29.2% in the experience set and from 15.5% to 26.5% in the evaluation set. Revisions also increase the frequency with which different requirements co-occur. The proportion of episodes with at least two additional requirement features rises from 20% to 59% in the experience

set and from 16% to 44% in the evaluation set. After all revisions, 17 experience episodes and nine evaluation episodes contain adjacency, temporal, and state requirements together. The proportion containing both state and temporal requirements increases from 11% to 48% and from 2% to 32%, respectively. Thus, revisions expand the goal set and increase the combinations of requirements that must be satisfied within the same episode.

Figure 3 presents the aggregate changes, proposition composition, and distribution of requirement combinations. Mean propositions per episode and episode percentages place the differently sized experience and evaluation sets on common scales.

Distribution of task-relevant entity categories. Following the style of PARTNR's entity-distribution visualizations (Chang et al., 2025), we report object, furniture, and room categories separately. Each proposition contributes one count to each category it mentions; multiple candidate instances of the same category within a proposition are counted once. Objects comprise movable objects referenced by placement or state propositions. Furniture comprises placement targets and furniture targeted by state operations. Rooms comprise explicit room targets and the scene regions containing placement-target furniture. Category mapping uses fixed aggregation rules: for example, dining-area models are counted as tables, and the compound laundry-room/mudroom label is grouped with laundry rooms.

Under this convention, object-category coverage grows from 70 to 79 categories in the experience set and from 50 to 63 in the evaluation set. Total object-category mentions increase from 352 to 639 and from 180 to 311, respectively. Furniture and room categories change less. This pattern is consistent with the construction procedure, which extends object and state requirements around existing scenes and task-relevant destinations. Figure 4 shows the object, furniture, and room distributions for both sets using horizontal stacked bars. Light segments count category mentions from original propositions retained in the final specification, and dark segments count mentions from propositions added by revisions and retained at the end. Each bar's total therefore equals the fully revised count. Deleted propositions contribute to neither segment; the complete initial counts, including subsequently deleted requirements, remain included in Table 14. Each panel independently sorts categories by its own fully revised counts in descending order. Object panels show the top 30 categories in each set; furniture and room panels show all categories present in that set's final specifications. Category coverage in the table is always computed from the complete data.

Composition of revision sequences. The experience and evaluation sets contain 181 and 91 retained events, comprising 281 and 122 atomic operations, respectively. An event can contain several atomic operations, and an operation can affect several propositions: for example, REORDER adds two placement propositions and their ordering relation. Events, operations, and propositions are therefore counted separately. The experience set adds 304 propositions and deletes 23, while the evaluation set adds 134 and deletes two.

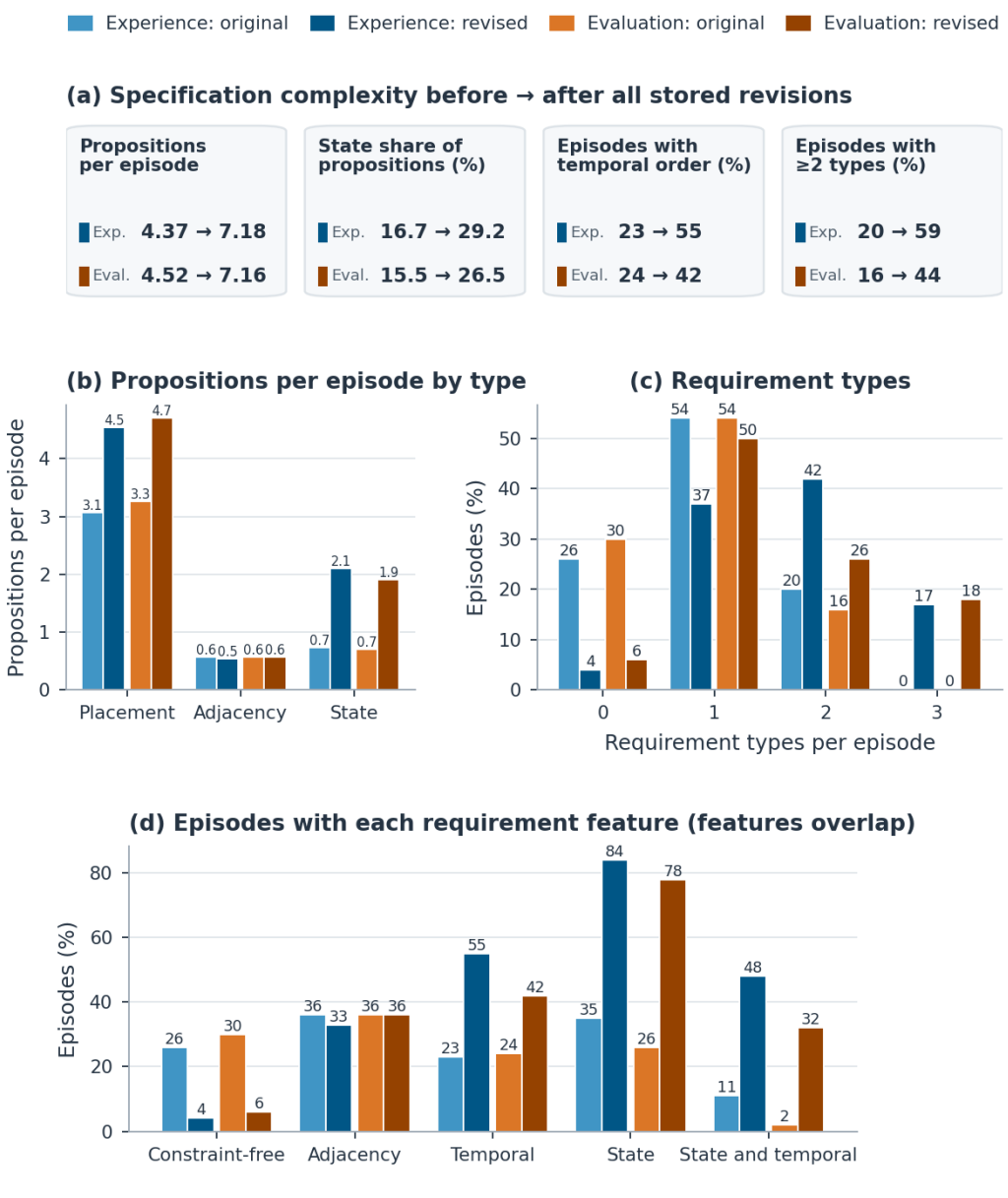


Figure 3: **Specification size and combinations of requirements before and after revisions.** Blue denotes the experience set and orange the evaluation set; light and dark shades indicate the initial and fully revised specifications. (a) Changes in key statistics. (b) Mean propositions per episode, divided into placement, adjacency, and state propositions. (c) The number of additional requirement features per episode. (d) Episode coverage of each feature and of the intersection of state and temporal requirements, with percentages labeled above the bars. The additional features are adjacency, temporal order, and object state. "Constraint-free" in (d) denotes episodes with none of these three features; such episodes can still contain placement goals and other evaluator constraints.

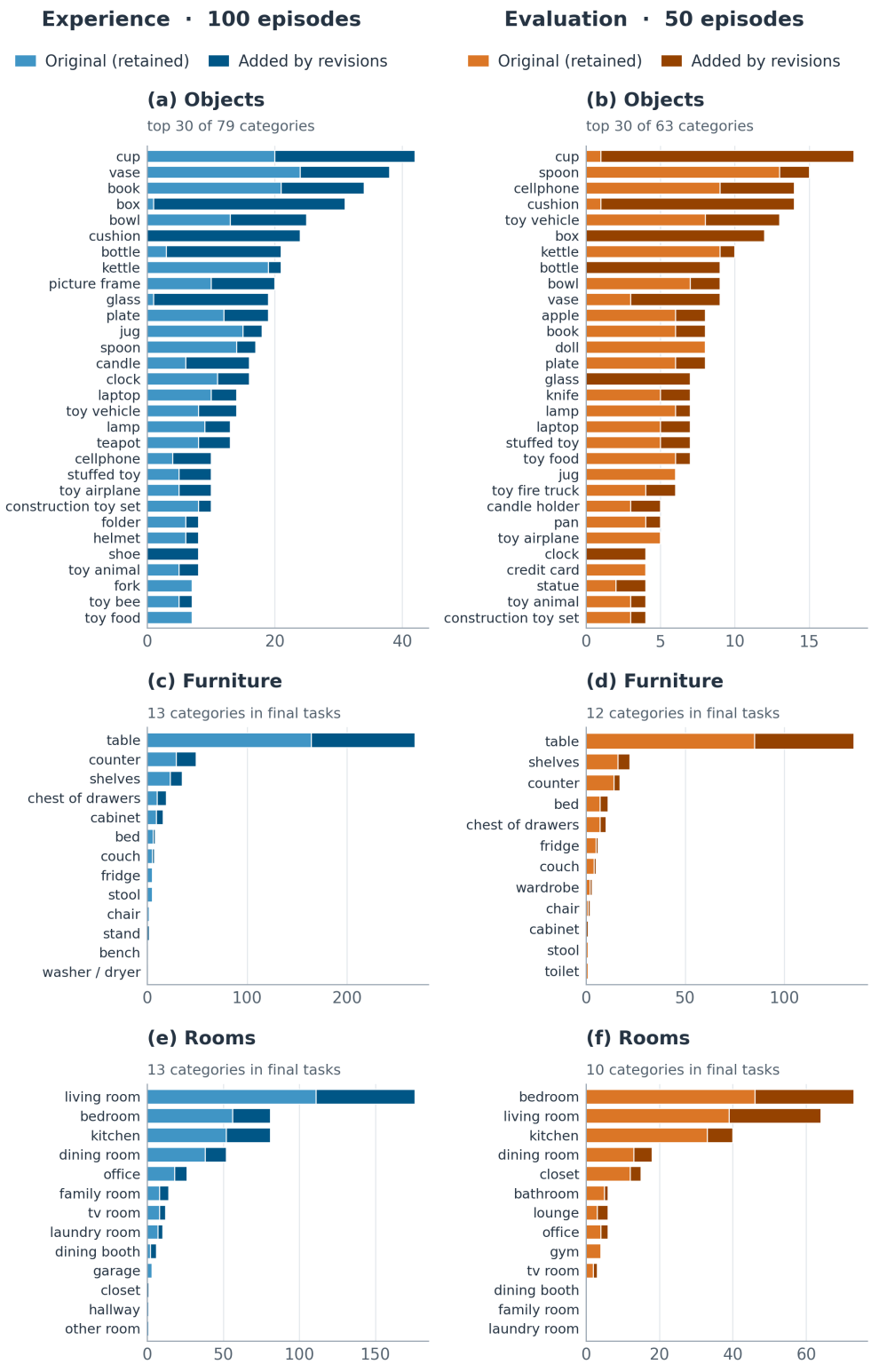


Figure 4: **Distributions of task-relevant object, furniture, and room categories.** Columns show 100 experience episodes (left) and 50 evaluation episodes (right). Light/dark segments count retained original/revision-added propositions, respectively, with each category counted once per proposition. Object panels show each set's top 30; furniture and room panels include all final categories. Panel (f) shows 13 room categories, including three with zero counts.

B TASK PROCESS AND EVALUATION SEMANTICS

B.1 TASK-EVOLVING PROCESS AND DECENTRALIZED DECISIONS

The team consists of a human agent H and a Spot robot S, both controlled by language-model planners. Let s_t denote the physical state, χ_t the high-level skills currently being executed by the two agents and their controller states, G_t the active task specification, η_t the evaluation history, $j(t)$ the number of revision events applied by time t, and Ξ_{run} the revision recipe used during execution. The evaluation history stores proposition results, the entity instances satisfying each proposition, and first-satisfaction indices, supporting the evaluation of historical dependencies, temporal order, and argument consistency. The augmented state is

$$X_t = (s_t, \chi_t, G_t, \eta_t, j(t), \Xi_{\mathrm{run}}) . \tag{19}$$

The agents plan independently and execute high-level skills asynchronously. Let $\mathcal{T}^a$ denote the planning times at which agent a can select its next skill. Its local history h_t^a contains its observations, actions and execution results, received task instructions, and coordination messages. Between planning times, the agent continues executing its current skill; a task revision neither cancels an ongoing skill nor changes existing object possession. After a revision batch, both agents receive the updated cumulative instruction, which an agent executing a skill uses when it next plans. Negotiated allocation plans provide advisory guidance, while each agent continues to select subsequent actions based on its own information. All protocols, including NEVER, receive the active task instruction. Each agent updates its local world graph through its own perception; its teammate's action results are not automatically propagated to that graph. The environment maintains the revision recipe, thresholds, and evaluation history without exposing them to the planners, which also lack access to the full physical state and future revisions. Action selection follows

$$u_t^a \sim \pi^a (\cdot \mid h_t^a) , \qquad t \in \mathcal{T}^a, \quad a \in \{H, S\}. \tag{20}$$

B.2 REVISION INJECTION AND STATE UPDATES

The physical state evolves as the agents execute their current skills. Using the active specification and existing history, the evaluator determines which propositions are eligible for evaluation, records their results, and updates completion. Revision checks use completion under the active task version, which need not increase monotonically as propositions and constraints change. Let $\mathcal{B}$ denote eligible check boundaries at which at least one agent has no high-level skill in progress, including the initial state, and let $K_{\mathrm{run}} = |\Xi_{\mathrm{run}}|$ be the number of events in the runtime recipe. At $t \in \mathcal{B}$, if j events have been applied, only the next event is checked. When $j < K_{\mathrm{run}}$ and $\mathrm{PC}_{G_t}(s_t, \eta_t) \geq \rho_{j+1}$, revision ξ_{j+1} is applied, the applied-event count is incremented, and the current state is re-evaluated under the updated specification. The next event is then checked, continuing until completion falls below its threshold or all events have been applied. Events applied consecutively at one boundary form a revision batch; added requirements can lower completion and leave subsequent events pending. After a nonempty batch, if execution must continue, the system delivers the updated cumulative instruction once to both agents and provides one negotiation opportunity. The protocol determines whether dialogue occurs and its scope. Each batch provides at most one opportunity, regardless of how many events or atomic operations it contains. Once all scheduled events have been applied and the active task succeeds, the episode terminates successfully without further negotiation.

Let d_j denote the realized injection time of event j. We index states by event to distinguish multiple updates within the same simulator step, using superscripts $-$ and $+$ for the states before application and after re-evaluation, respectively. The task transformation ξ_j jointly updates the instruction, propositions, dependencies, and constraints. The history migration operator Ψ_{ξ_j} removes records for cancelled propositions, remaps retained records to the updated proposition indices, and initializes historical entries for new propositions. The operation EvalUpdate then uses the updated specification, current physical state, and migrated history to determine evaluation eligibility, append the current results, and update first-satisfaction indices and completion. Because some dependencies refer to earlier evaluation results, immediate re-evaluation depends on both the physical state and the history. Injection preserves object locations, object states, and agent possession, and does not interrupt ongoing skills. A new proposition may be satisfied during this re-evaluation, but its satisfaction record starts only upon introduction. Multiple events may share an injection time, with history migration and re-evaluation performed in stored order. For a given task under ONLINE-REVISED, all

models and negotiation protocols use the same stored recipe: revision content, order, and thresholds remain fixed, while realized injection times and triggered-event counts depend on the execution trajectory. Events whose thresholds are not reached are not forcibly applied, and applied events are not revoked if completion subsequently decreases. A revision updates the state as

$$\begin{aligned} G_j^+ &= \xi_j(G_j^-), \\ (s_j^+, \chi_j^+) &= (s_j^-, \chi_j^-), \\ \widetilde{\eta}_j &= \Psi_{\xi_j}(\eta_j^-), \\ \eta_j^+ &= \mathsf{EvalUpdate}\left(G_j^+, s_j^+, \widetilde{\eta}_j\right). \end{aligned} \tag{21}$$

B.3 Three Task Conditions

The three task conditions share the initial physical state s_0 of each base task, agent capabilities, observation interfaces, and execution budget. Let $\mathcal{G}_0$ be the original specification and $\Xi = ((\rho_1, \xi_1), \ldots, (\rho_K, \xi_K))$ the complete stored recipe. Applying all revisions in order yields the final specification $\mathcal{G}_K = \xi_K \circ \cdots \circ \xi_1(\mathcal{G}_0)$; when $K = 0$, the final and original specifications coincide. ORIGINAL-STATIC uses the original specification throughout execution. FINAL-UPFRONT constructs the final specification before execution and provides all final requirements at episode start, with no revisions injected during execution. ONLINE-REVISED starts from the original specification and applies stored events according to progress; its active version after event j is $\mathcal{G}_j$. The initial active specifications and runtime recipes are summarized below.

Table 15: Initial active specifications, runtime revision recipes, and intended final specifications under the three task conditions.

Condition	Initial active specification	Runtime revision recipe	Intended final specification
ORIGINAL-STATIC	$\mathcal{G}_0$	$\varnothing$	$\mathcal{G}_0$
FINAL-UPFRONT	$\mathcal{G}_K$	$\varnothing$	$\mathcal{G}_K$
ONLINE-REVISED	$\mathcal{G}_0$	Ξ	$\mathcal{G}_K$

Each condition maintains a separate evaluation history. For condition c, let G^c_{init} denote its initial active specification and Init_c its evaluator initialization operation. The initial specification determines the propositions and constraints associated with that history. The conditions therefore share the physical initial state but differ in their augmented initial states, which also contain the task specification, evaluation history, and runtime recipe. The comparison between ORIGINAL-STATIC and FINAL-UPFRONT characterizes performance changes associated with final-task expansion. FINAL-UPFRONT and ONLINE-REVISED match intended final requirements and compare performance when those requirements are available from the outset or introduced incrementally during execution. In ONLINE-REVISED, the physical state and evaluation history accumulate across intermediate task versions, so execution is shaped jointly by new requirements and the consequences of earlier actions. The initialized evaluation history is denoted by

$$\eta_0^c = \mathsf{Init}_c\left(s_0, G^c_{\text{init}}\right). \tag{22}$$

B.4 History-Aware Completion and Termination

History preservation and new requirements. Retained propositions preserve their historical evidence and first-satisfaction indices after re-indexing; deleting a proposition also updates its historical records and associated references. A new proposition is recorded as unsatisfied in all earlier history rows, with its first-satisfaction index initially unset. Let ι_i denote the evaluation-history index at which proposition i is introduced, and let τ_i^{sat} denote the index at which it is first recorded as satisfied, taking the value -1 if it has never been satisfied. These indices identify records in the evaluation history, not simulator steps. A physical state established before a revision may satisfy a new requirement at its first evaluation after introduction, but its first-satisfaction record is not backdated:

$$\tau_i^{\text{sat}} \in \{-1\} \cup \{\iota_i, \iota_i + 1, \ldots\}. \tag{23}$$

Completion. Suppose the active task contains $n(G)$ propositions indexed by $0, \ldots, n(G)-1$, and let $v_i \in \{0,1\}$ indicate whether the satisfaction record of proposition i passes its relevant constraints under the current state and history. Dependencies determine whether a proposition is eligible for evaluation at a particular evaluation step; temporal, terminal-satisfaction, same-argument, and different-argument constraints determine the validity of its satisfaction record. Terminal-satisfaction constraints check the latest evaluation state during online scoring and the final state at episode termination, allowing a requirement to become temporarily unsatisfied and subsequently be restored. For a temporal constraint, if the relevant propositions and temporal edge are retained and their first-satisfaction records are not reset, an incorrect first-satisfaction order cannot be repaired by subsequent re-execution. Complete-sequence validation establishes the existence of successful executions under both successive revisions and upfront final requirements, although individual model trajectories may still violate temporal order. Initial proposition sets are nonempty, and valid deletions retain at least one proposition, so every active specification satisfies $n(G) \geq 1$. Completion and active-task success are defined as

$$\begin{aligned} \mathrm{PC}_G(s,\eta) &= \frac{1}{n(G)} \sum_{i=0}^{n(G)-1} \mathbf{1}\left[\tau_i^{\mathrm{sat}} \geq 0\right] v_i, \\ \mathrm{Success}_G(s,\eta) &= \mathbf{1}\left[\mathrm{PC}_G(s,\eta) = 1\right]. \end{aligned} \tag{24}$$

Scoring and termination. Active-task success means that PC equals one. Automatic successful termination additionally requires all events in the runtime recipe to have been applied, i.e., $j(t) = K_{\mathrm{run}}$. This additional condition holds from episode start in the two static conditions, whose runtime recipes are empty. All conditions use the same budget of $20{,}000$ simulator steps; budget-limited outcomes are scored using the specification active at termination and its evaluation history. If an ONLINE-REVISED episode exhausts its budget with pending events, its terminal score refers to an intermediate specification, whereas FINAL-UPFRONT always uses the complete final specification. Their PC values may consequently involve different proposition sets and constraints. The three-condition design thus matches intended final requirements without guaranteeing identical realized terminal specifications for all budget-limited trajectories. SR averages the active-task success indicators at episode termination. For N evaluation episodes, with t_e^{end} denoting the termination time of episode e,

$$\mathrm{SR} = \frac{1}{N} \sum_{e=1}^{N} \mathrm{Success}_{G^e_{t_e^{\mathrm{end}}}}\left(s^e_{t_e^{\mathrm{end}}}, \eta^e_{t_e^{\mathrm{end}}}\right). \tag{25}$$

Decomposing completion changes. A revision can change completion without changing the physical state. Suppose an episode applies $m = j(T)$ events. Let p_j^- and p_j^+ denote PC immediately before event j and after its re-evaluation, respectively, and define the revision-induced evaluation jump as $J_j = p_j^+ - p_j^-$. Set $p_0^+ = p_{\mathrm{init}}$ to completion under the condition's initial specification before any online revision, and $p_{m+1}^- = p_{\mathrm{end}}$ to completion at termination. Let Δ_ℓ denote the completion change within a segment of fixed specification between revisions. Its two endpoints use the same active specification, whereas J_j compares the specifications immediately before and after one revision. Endpoints are indexed by event, allowing separate changes to be computed even when multiple events share a simulator step. If no physical execution or additional evaluation update occurs between consecutive events in a batch, the intervening segment has zero completion change. For the two static conditions, $m = 0$, and the decomposition reduces to the completion change over the entire fixed-task episode. Segment returns used for experience evaluation compute completion increments under a common task specification, separating the revision-induced jumps J_j from execution progress following the selected negotiation mode. The segment increments and total change satisfy

$$\Delta_\ell = p_{\ell+1}^- - p_\ell^+, \qquad \ell = 0, \ldots, m. \tag{26}$$

$$p_{\mathrm{end}} - p_{\mathrm{init}} = \sum_{\ell=0}^{m} \Delta_\ell + \sum_{j=1}^{m} J_j. \tag{27}$$

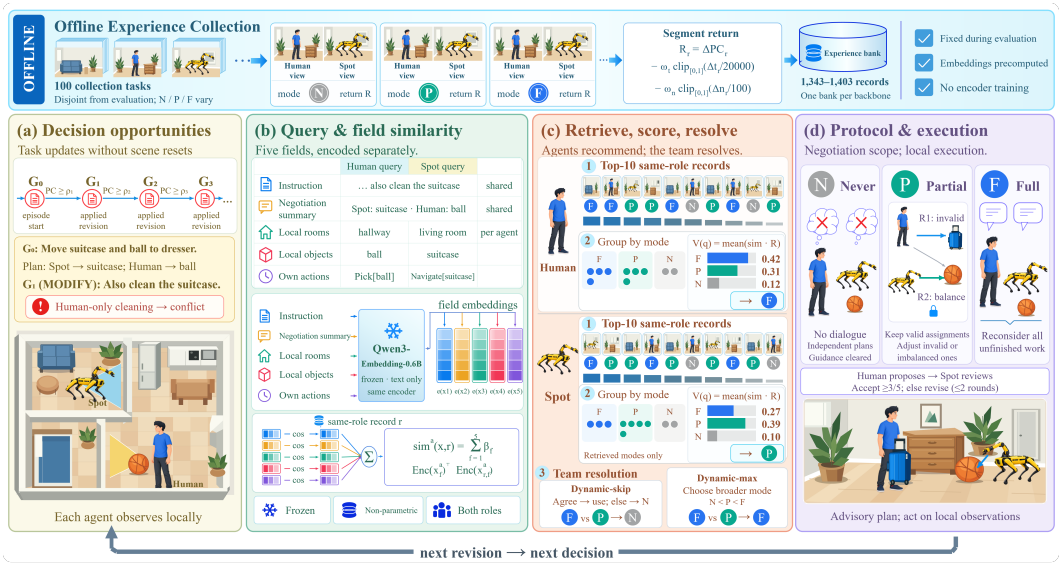


Figure 5: **Experience-based negotiation-mode selection.** The upper panel shows offline experience collection and bank construction; the bank remains fixed during evaluation. (a) Mode decisions occur at episode start and after each revision batch takes effect, while the physical scene persists. (b) A frozen text encoder separately encodes the five query fields, and a weighted sum of field-wise cosine similarities measures contextual similarity. (c) Each role retrieves the top-k same-role records, groups them by executed mode, and recommends a mode using the group mean of similarity–return products. In the illustrated Full–Partial disagreement, Dynamic-skip selects Never and Dynamic-max selects Full. (d) The selected mode determines the negotiation scope and allocation guidance, after which each agent executes actions based on its local information.

C Experience-Based Adaptive Selection

Dynamic selects among Never, Partial, and Full, denoted by N, P, and F, according to the current task and local execution context. Each agent retrieves historical experience for its own role and combines contextual similarity with execution returns to recommend a mode. A team resolution rule then combines the two recommendations. Figure 5 summarizes offline experience collection and the online process from contextual retrieval to team mode selection and decentralized execution. Table 16 reports the experience-bank sizes for the four backbones and the selector parameters.

C.1 Experience collection and segment returns

Experience is collected on a separate set of 100 tasks whose composite scene–episode identifiers do not overlap with those of the 50 evaluation tasks. Each of the four backbones collects and uses its own experience bank, and Dynamic-skip and Dynamic-max share the same bank within each backbone. Collection and evaluation use identical negotiation and execution prompts. Approximately balanced mode schedules cover decision opportunities at episode start and after task revisions. For a task e with $c_e \in \{0, 1, 2, 3\}$ candidate thresholds, collection assigns $3(c_e + 1)$ mode sequences, corresponding to 3, 6, 9, or 12 runs. The schedules are determined by random seed 42 and the task identifier. The candidate-threshold count can exceed the number of finalized revision events, while episode termination and revision batching affect the number of recorded decisions.

The bank sizes in Table 16 count decision records. Each record contains the contexts of the human agent and the Spot robot, one executed team mode, and a shared segment return. During collection, the contexts are cached before mode selection; the return is appended and the record saved after the subsequent execution segment ends. For record r, the segment extends from the mode decision to the next task revision or episode termination. The completion increment $\Delta\mathrm{PC}_r$ follows the definition of Δ_ℓ in Eq. (26), with the same task specification at both endpoints. If the segment ends at a revision, its terminal PC is measured immediately before that revision takes effect. Let Δt_r denote

the increment in simulator steps and Δn_r the increment in the sum of both agents' execution-planner calls. The segment return is

$$R_r = \Delta \mathrm{PC}_r - \omega_t \ \mathrm{clip}_{[0,1]}\left(\frac{\Delta t_r}{20000}\right) - \omega_n \ \mathrm{clip}_{[0,1]}\left(\frac{\Delta n_r}{100}\right). \tag{28}$$

Here, ω_t and ω_n are the penalty weights for simulator steps and team execution-planner calls, respectively, with values reported in Table 16. The operator $\mathrm{clip}_{[0,1]}$ clips its input to $[0, 1]$. Both cost terms use within-segment counter increments before normalization and clipping. The return includes no direct penalty for negotiation tokens or retrieval latency.

C.2 Role-conditioned retrieval

At each mode-selection opportunity, the query for role $a \in \{H, S\}$ consists of five fields:

$$x^a = \left(I, D^{\mathrm{hist}}, \mathrm{Rooms}^a, \mathrm{Objects}^a, \mathrm{History}^a\right). \tag{29}$$

Here, I is the current task instruction and D^{hist} summarizes prior negotiation; both are shared across roles. Rooms, objects, and action histories reflect each agent's local information. Object records include the room and supporting furniture, and state-change outcomes can also appear in the action history. All fields are taken from the context available before the current mode decision. The query does not explicitly encode a structured task difference or a separate indicator of whether historical allocation guidance remains active.

A frozen Qwen3-Embedding-0.6B encoder separately encodes and normalizes each field. Strings are encoded directly, while structured fields are serialized as JSON with sorted keys. Let $\mathrm{Enc}(\cdot)$ denote the field encoder with unit-normalized outputs, $x^a_{r,f}$ the f-th field for role a in record r, and β_f the corresponding field weight, whose value is reported in Table 16. Query-to-record similarity is the weighted sum of the five field-wise cosine similarities:

$$\mathrm{sim}^a(x, r) = \sum_{f=1}^{5} \beta_f \ \mathrm{Enc}(x^a_f)^\top \mathrm{Enc}(x^a_{r,f}). \tag{30}$$

Each role retrieves the top-k most similar same-role records from the current backbone's experience bank and then groups them by executed mode. Let $\mathcal{U}^a_q$ denote the subset associated with mode $q \in \{N, P, F\}$. The empirical mode score and the role's recommendation are

$$\widehat{V}^a(q) = \frac{1}{|\mathcal{U}^a_q|} \sum_{r \in \mathcal{U}^a_q} \mathrm{sim}^a(x, r) R_r, \qquad q_a \in \operatorname*{arg\,max}_{q:\, |\mathcal{U}^a_q| > 0} \widehat{V}^a(q). \tag{31}$$

The score is the arithmetic mean of similarity–return products within each group, with the record count as its denominator. The retrieval depth k applies to the total number of records per role before mode grouping. Modes absent from the retrieved set are not scored, and no additional minimum-support requirement or similarity threshold is imposed. The retrieval depth, five field weights, return penalty weights, and normalization scales were all specified in advance without tuning on the 50 evaluation tasks. Experience records, stored returns, and bank embeddings remain fixed throughout evaluation.

C.3 Team mode resolution and computational cost

Dynamic-skip and Dynamic-max use different rules to combine the recommendations (q_H, q_S) of the human agent and the Spot robot into a team mode. Dynamic-skip adopts the recommended mode when the two agents agree and selects Never otherwise, including when they disagree between Full and Partial. Dynamic-max selects the broader mode under $N < P < F$, comparing mode scope rather than the agents' numerical scores. Both rules are symmetric in the roles, with the complete mapping given by

(q_H, q_S)	(N,N)	(P,P)	(F,F)	(N,P)	(N,F)	(P,F)
$\mathcal{R}_{\mathrm{skip}}$	N	P	F	N	N	N
$\mathcal{R}_{\max}$	N	P	F	P	F	F

. (32)

After the team mode is selected, each agent continues to choose physical actions based on its own information. Selecting Never clears the active allocation plan and the Partial baseline, while the historical negotiation summary can remain in the retrieval context. If Partial is selected again before a new allocation plan has been installed, its frozen set is empty and no prior assignments are required to be retained. Negotiation and execution still use the Partial prompts.

Mode selection requires no additional generative language-model call or parameter update; its online cost comes from query encoding and experience retrieval. Selecting Full or Partial still incurs language-model calls for the negotiation dialogue. Experience collection and bank-embedding construction incur offline costs. The action-planning costs reported in the main text cover execution planning and exclude mode-selection and negotiation costs.

Table 16: **Experience-bank sizes and adaptive-selection parameters.** Within each backbone, the two Dynamic configurations share the bank collected by that backbone. Selector parameters are identical across all evaluated backbones.

Component	Setting
DeepSeek-v4-flash experience bank	1,403 decision records
Devstral-2-123B experience bank	1,343 decision records
Gemma-4-31B experience bank	1,375 decision records
GPT-OSS-20B experience bank	1,395 decision records
Embedding model	Frozen Qwen3-Embedding-0.6B
Retrieval depth (top-k)	$k = 10$
Field similarity weight	$\beta_f = 0.2$
Mode-score denominator	Retrieved record count for the mode
Team resolution rules	Never on disagreement; or maximum under $N < P < F$
Simulator-step penalty weight	$\omega_t = 0.2$
Team planner-call penalty weight	$\omega_n = 0.1$
Return normalization scales	20,000 simulator steps; 100 execution-planner calls summed across both agents

D Reported Measurements and Statistical Specification

D.1 Never baseline measurements

Tables 17–21 report results under the Never protocol for all nine backbones across the Original-Static, Final-Upfront, and Online-Revised conditions, using the same 50 evaluation tasks. These measurements support the task-condition comparison in Section 4 and provide baselines for the negotiation comparisons in Section 5.

Planning cycles denote mean execution-planner calls per agent, averaged per episode, i.e. half the sum used by the return in Eq. 28. Token totals cover action planning only and therefore remain comparable across conditions.

Table 23 summarizes cross-condition effort ratios computed from the nine-backbone model-equal means.

D.2 Full results across negotiation configurations on the revised tasks

Table 24 reports complete five-configuration summaries for the four backbones, and Tables 25–28 break them down by constraint type. All token columns report action-planning usage by the two execution agents and exclude negotiation.

The two Dynamic modes. Negotiation opportunities arise at episode start and whenever a revision is applied. At each opportunity the human agent and the Spot agent independently select a negotiation level from their own experience bases: none, partial, or full. When the two selections differ, a rule is required to reduce them to a single executed negotiation, and Dynamic-skip and

Table 17: Reported measurements on the original task condition under Never. SR and PC are fractions; steps and mean per-agent planner calls are reported averages. Input, Output, and Total report action-planning token usage.

Model	Sim steps	SR	PC	Calls / agent	Input	Output	Total
GPT-4o mini	5904.640	0.720	0.8731	18.055	148451	2506	150957
DeepSeek-v4-flash	4293.760	0.820	0.9284	16.225	145798	3396	149194
GPT-OSS-20B	5198.790	0.810	0.9048	15.035	118567	26749	145316
Gemma-4-31B	4418.950	0.850	0.9421	12.810	105847	2111	107958
Qwen3-30B-A3B	7445.620	0.670	0.8359	21.360	204894	3703	208597
Devstral-2-123B	4805.270	0.820	0.9299	15.650	132580	2589	135169
Llama-3.3-70B	3950.440	0.810	0.9226	13.980	117782	3786	121568
GPT-OSS-120B	3452.000	0.920	0.9741	11.510	79004	11883	90887
GPT-4.1 nano	9651.150	0.160	0.3653	38.615	419528	7013	426541

Table 18: Performance and planning cost under Original-Static with the NEVER protocol, stratified by task-constraint dimension and averaged equally across nine backbones.

Constraint	Presence	n	Sim steps ↓	SR (%) ↑	PC (%) ↑	Planning cycles	Tokens		
							Input	Output	Total
Temporal	Absent	29	4924.184	75.86	86.65	16.398	135817	5718	141535
	Present	21	6194.809	69.31	83.41	20.540	201980	8966	210946
Object state	Absent	11	4732.354	76.26	88.39	17.730	164442	6586	171028
	Present	39	5662.473	72.22	84.41	18.253	163370	7222	170592
Spatial adjacency	Absent	32	5959.872	69.45	84.67	19.059	172606	7495	180101
	Present	18	4565.358	79.63	86.40	16.500	147606	6348	153954

Note. Metrics are averaged over two runs within each model and subgroup, then equally across the nine models. n counts distinct tasks per model in each subgroup, each evaluated twice. The three constraint dimensions are analyzed separately and may overlap. Planning cycles is the mean count per agent. SR and PC are percentages. Tokens cover action planning by both agents; Input and Output are rounded separately to integers, and Total is their sum.

Dynamic-max instantiate the two natural choices. Dynamic-skip requires agreement: if the selections differ, the two agents do not communicate at that opportunity. Dynamic-max takes the higher of the two levels, so a partial or full selection by either agent is executed at the corresponding level. Full and Partial are fixed configurations in which both agents negotiate at the preset level at every opportunity, and no reduction applies.

Table 19: Reported measurements on the merged-instruction task condition under Never (all revision instructions collapsed into a single final instruction applied at episode start). SR and PC are fractions; steps and mean per-agent planner calls are reported averages. Input, Output, and Total report action-planning token usage.

Model	Sim steps	SR	PC	Calls / agent	Input	Output	Total
GPT-4o mini	12508.780	0.310	0.7667	31.275	296625	4117	300742
DeepSeek-v4-flash	7901.700	0.630	0.9006	25.865	274299	5432	279731
GPT-OSS-20B	9346.610	0.580	0.8706	29.310	272168	54512	326680
Gemma-4-31B	8468.330	0.660	0.9243	22.515	220144	3971	224115
Qwen3-30B-A3B	12061.900	0.390	0.7489	34.195	380451	6915	387366
Devstral-2-123B	8245.930	0.620	0.8941	25.845	270082	4909	274991
Llama-3.3-70B	6676.150	0.620	0.8997	23.085	237792	5537	243329
GPT-OSS-120B	6104.800	0.800	0.9419	21.135	184710	29694	214404
GPT-4.1 nano	11085.210	0.070	0.2974	43.285	487453	8227	495680

Table 20: Performance and planning cost under Final-Upfront with the NEVER protocol, stratified by task-constraint dimension and averaged equally across nine backbones.

Constraint	Presence	n	Sim steps ↓	SR (%) ↑	PC (%) ↑	Planning cycles	Tokens		
							Input	Output	Total
Temporal	Absent	29	7811.349	62.26	84.78	24.952	236257	11219	247476
	Present	21	11011.685	37.83	74.57	33.402	367847	17131	384978
Object state	Absent	11	7191.616	62.12	83.77	22.755	232791	9773	242564
	Present	39	9709.403	49.15	79.56	30.122	308091	14810	322901
Spatial adjacency	Absent	32	9369.047	48.79	79.81	28.628	295153	13419	308572
	Present	18	8775.833	57.72	81.69	28.276	285075	14204	299279

Note. Each metric is the arithmetic mean of the nine models' subgroup-specific two-file averages, with equal model weights and no episode-count weighting. n denotes the number of tasks per model in each subgroup. Each constraint dimension is analyzed separately. SR and PC are expressed as percentages. Input and Output tokens are rounded separately to integers; Total is their sum.

Table 21: Reported measurements on the revised task condition under Never. SR and PC are fractions; steps and mean per-agent planner calls are reported averages. Input, Output, and Total report action-planning token usage.

Model	Sim steps	SR	PC	Calls / agent	Input	Output	Total
GPT-4o mini	13190.890	0.280	0.7217	32.510	319893	4710	324603
DeepSeek-v4-flash	8302.125	0.610	0.8937	26.148	276348	5857	282205
GPT-OSS-20B	9975.630	0.550	0.8317	30.005	279222	54987	334209
Gemma-4-31B	8632.350	0.620	0.9053	22.695	233136	4325	237461
Qwen3-30B-A3B	13101.520	0.310	0.7446	35.315	396018	6487	402505
Devstral-2-123B	8975.520	0.520	0.8558	25.340	264153	4909	269062
Llama-3.3-70B	8140.220	0.540	0.8805	25.185	261049	5877	266926
GPT-OSS-120B	7737.360	0.700	0.9119	24.430	215623	30809	246432
GPT-4.1 nano	12504.480	0.040	0.2727	40.860	438038	7261	445299

Table 22: Performance and planning cost by task-constraint dimension under Never on online-revised tasks, averaged equally across nine models.

Constraint	Presence	n	Sim steps ↓	SR (%) ↑	PC (%) ↑	Calls / agent	Tokens		
							Input	Output	Total
Temporal	Absent	29	8982.604	56.23	80.66	25.791	239046	10891	249937
	Present	21	11553.148	32.67	74.27	33.825	379804	18088	397892
Object state	Absent	11	6723.187	59.34	83.33	22.415	223729	10537	234266
	Present	39	11004.015	42.66	76.46	31.069	319159	14866	334025
Spatial adjacency	Absent	32	9824.440	45.49	77.37	28.312	286729	13033	299762
	Present	18	10484.975	47.84	79.05	30.681	318494	15478	333972

Note. Each metric is the arithmetic mean of the nine model-specific subgroup estimates, with equal model weights rather than episode-count weighting. Within-model estimates average the two result files, except for DeepSeek-v4-flash, which uses the mean of four runs. n denotes the number of episodes per model in each subgroup. Each constraint dimension defines a separate absent/present split; subsets may overlap across dimensions. SR and PC are expressed as percentages; Calls / agent denotes mean per-agent planner calls. Input, Output, and Total report action-planning token usage. Input and Output are rounded separately to integers; Total is their sum.

Table 23: **Aggregate execution-effort ratios across the three conditions under NEVER.** Ratios are computed after equally averaging per-episode means across the nine backbones. Planner calls are averaged across the two agents; action-planning tokens sum both agents' input and output usage and exclude allocation dialogue.

Condition comparison	Steps	Planner calls	Action tokens
Final-Upfront / Original-Static (B/A)	1.677	1.571	1.788
Online-Revised / Original-Static (C/A)	1.844	1.608	1.828
Online-Revised / Final-Upfront (C/B)	1.099	1.023	1.022

Table 24: **Complete quality, execution, and token summaries for five configurations.** SR and PC are percentages. Steps and planning calls are episode averages, with calls averaged across the two agents. Token columns report action-planning input and output usage per episode. Entries are equally weighted means over the runs available for each model and configuration (four runs for the DeepSeek Never and Full rows, two runs elsewhere).

Model	Configuration	Steps	SR (%)	PC (%)	Planning calls (agent mean)	Input	Output	Total
DeepSeek-v4-flash	Never	8,302.125	61.0	89.37	26.148	276,348	5,857	282,205
	Full	7,598.215	61.5	86.21	28.615	324,678	5,830	330,508
	Partial	7,777.520	63.0	88.74	22.760	242,853	4,208	247,061
	D-skip	7,683.580	66.0	89.27	24.830	262,381	4,706	267,087
	D-max	7,667.540	67.0	88.54	24.710	261,236	4,684	265,920
Gemma-4-31B	Never	8,632.350	62.0	90.53	22.695	233,136	4,325	237,461
	Full	7,976.320	68.0	92.17	22.245	242,808	4,323	247,131
	Partial	7,588.990	71.0	92.48	21.940	221,161	3,437	224,598
	D-skip	8,263.290	69.0	91.79	21.940	218,352	3,990	222,342
	D-max	7,404.640	75.0	91.80	20.470	204,013	3,726	207,739
GPT-OSS-20B	Never	9,975.630	55.0	83.17	30.005	279,222	54,987	334,209
	Full	9,772.540	55.0	82.91	27.515	272,458	44,534	316,992
	Partial	8,394.620	57.0	83.95	24.510	221,375	40,672	262,047
	D-skip	8,717.910	62.0	86.23	27.050	251,238	42,078	293,316
	D-max	9,246.980	56.0	84.16	27.760	266,777	51,786	318,563
Devstral-2-123B	Never	8,975.520	52.0	85.58	25.340	264,153	4,909	269,062
	Full	9,633.160	48.0	78.08	30.490	360,881	6,259	367,140
	Partial	7,854.190	46.0	75.04	29.580	317,265	4,904	322,169
	D-skip	9,307.740	53.0	82.95	28.090	312,251	5,496	317,747
	D-max	9,336.030	52.0	82.21	27.605	310,467	5,208	315,675

Table 25: **Per-condition breakdown for DeepSeek-v4-flash under five configurations.** Each column pair splits the same 50 episodes by whether a constraint of that type is present; the three splits are independent attributes and are not mutually exclusive. Entries are equally weighted means over the available runs for each configuration (four runs for Never and Full, two for Partial, D-skip, and D-max). Tokens are action-planning input plus output, in thousands.

Metric	Configuration	Temporal		Object state		Spatial	
		Absent $n=29$	Present $n=21$	Absent $n=11$	Present $n=39$	Absent $n=32$	Present $n=18$
Success rate (%)	Never	71.6	46.4	65.9	59.6	57.8	66.7
	Full	75.9	41.7	61.4	61.5	57.8	68.1
	Partial	74.1	47.6	68.2	61.5	59.4	69.4
	D-skip	74.1	54.8	81.8	61.5	67.2	63.9
	D-max	77.6	52.4	72.7	65.4	65.6	69.4
Percent complete (%)	Never	91.5	86.5	91.0	88.9	89.0	90.1
	Full	88.6	82.9	89.8	85.2	84.5	89.3
	Partial	90.8	85.9	92.7	87.6	87.2	91.5
	D-skip	91.6	86.0	96.6	87.2	90.7	86.7
	D-max	91.4	84.6	92.8	87.4	88.2	89.1
Simulation steps	Never	6610	10639	5987	8955	8292	8320
	Full	6331	9348	5796	8107	7824	7198
	Partial	5939	10316	6043	8267	7774	7785
	D-skip	6602	9177	6118	8125	7483	8040
	D-max	5795	10253	6208	8079	7646	7705
Planning calls (agent mean)	Never	22.8	30.7	20.2	27.8	25.1	28.0
	Full	25.3	33.2	26.5	29.2	29.0	27.9
	Partial	18.3	29.0	19.7	23.6	22.3	23.5
	D-skip	21.9	28.8	17.9	26.8	24.8	24.9
	D-max	20.5	30.5	19.4	26.2	23.9	26.2
Action-planning tokens (K)	Never	221	366	210	302	268	308
	Full	268	416	297	340	329	334
	Partial	177	344	212	257	242	256
	D-skip	214	340	176	293	265	271
	D-max	198	360	197	285	255	285

Table 26: **Per-condition breakdown for Gemma-4-31B under five configurations.** Each column pair splits the same 50 episodes by whether a constraint of that type is present; the three splits are independent attributes and are not mutually exclusive. All entries are two-run means over the episodes in the corresponding subset. Tokens are action-planning input plus output, in thousands.

Metric	Configuration	Temporal		Object state		Spatial	
		Absent $n=29$	Present $n=21$	Absent $n=11$	Present $n=39$	Absent $n=32$	Present $n=18$
Success rate (%)	Never	72.4	47.6	68.2	60.3	57.8	69.4
	Full	81.0	50.0	59.1	70.5	60.9	80.6
	Partial	77.6	61.9	72.7	70.5	57.8	94.4
	D-skip	77.6	57.1	72.7	68.0	62.5	80.6
	D-max	82.8	64.3	81.8	73.1	65.6	91.7
Percent complete (%)	Never	93.4	86.6	87.7	91.3	88.8	93.7
	Full	94.8	88.6	89.7	92.9	90.2	95.7
	Partial	94.4	89.8	93.2	92.3	88.5	99.6
	D-skip	93.6	89.3	93.5	91.3	89.7	95.4
	D-max	94.6	87.9	93.7	91.3	89.5	96.0
Simulation steps	Never	7088	10765	6067	9356	8688	8534
	Full	5833	10937	6464	8403	8391	7238
	Partial	6263	9420	6342	7941	8936	5194
	D-skip	6597	10564	6064	8884	8758	7384
	D-max	5421	10144	6853	7560	8246	5909
Planning calls (agent mean)	Never	19.4	27.3	15.3	24.8	22.1	23.8
	Full	18.4	27.5	15.9	24.0	22.4	22.0
	Partial	18.5	26.8	15.8	23.7	23.7	18.8
	D-skip	18.3	27.0	17.1	23.3	22.9	20.2
	D-max	16.9	25.4	17.1	21.4	21.5	18.6
Action-planning tokens (K)	Never	182	314	150	262	230	251
	Full	184	334	183	265	247	248
	Partial	169	301	154	244	248	184
	D-skip	168	298	175	236	234	201
	D-max	154	282	174	217	223	181

Table 27: **Per-condition breakdown for GPT-OSS-20B under five configurations.** Each column pair splits the same 50 episodes by whether a constraint of that type is present; the three splits are independent attributes and are not mutually exclusive. All entries are two-run means over the episodes in the corresponding subset. Tokens are action-planning input plus output, in thousands.

Metric	Configuration	Temporal		Object state		Spatial	
		Absent $n=29$	Present $n=21$	Absent $n=11$	Present $n=39$	Absent $n=32$	Present $n=18$
Success rate (%)	Never	69.0	35.7	59.1	53.9	54.7	55.6
	Full	72.4	31.0	59.1	53.9	54.7	55.6
	Partial	67.2	42.9	54.6	57.7	57.8	55.6
	D-skip	77.6	40.5	50.0	65.4	57.8	69.4
	D-max	72.4	33.3	59.1	55.1	56.3	55.6
Percent complete (%)	Never	88.9	75.2	80.2	84.0	82.4	84.6
	Full	90.3	72.7	82.1	83.2	82.6	83.5
	Partial	88.5	77.7	78.3	85.6	84.0	83.9
	D-skip	91.9	78.4	83.0	87.2	85.2	88.0
	D-max	90.3	75.7	80.3	85.2	84.5	83.5
Simulation steps	Never	8409	12139	7612	10642	9147	11449
	Full	7114	13445	7575	10392	9338	10546
	Partial	6831	10554	6569	8910	7519	9952
	D-skip	6233	12150	7258	9130	8543	9028
	D-max	7578	11552	5189	10392	8619	10364
Planning calls (agent mean)	Never	23.6	38.8	26.2	31.1	28.2	33.2
	Full	22.3	34.7	18.5	30.1	26.6	29.2
	Partial	20.0	30.8	21.5	25.4	22.3	28.4
	D-skip	20.5	36.1	22.5	28.3	26.3	28.3
	D-max	22.6	34.9	20.1	29.9	26.4	30.2
Action-planning tokens (K)	Never	229	479	295	345	313	372
	Full	237	428	191	353	302	343
	Partial	191	361	236	270	233	314
	D-skip	200	422	249	306	288	303
	D-max	233	437	236	342	302	348

Table 28: **Per-condition breakdown for Devstral-2-123B under five configurations.** Each column pair splits the same 50 episodes by whether a constraint of that type is present; the three splits are independent attributes and are not mutually exclusive. All entries are two-run means over the episodes in the corresponding subset. Tokens are action-planning input plus output, in thousands.

Metric	Configuration	Temporal		Object state		Spatial	
		Absent $n=29$	Present $n=21$	Absent $n=11$	Present $n=39$	Absent $n=32$	Present $n=18$
Success rate (%)	Never	63.8	35.7	72.7	46.2	56.3	44.4
	Full	58.6	33.3	54.6	46.2	45.3	52.8
	Partial	63.8	21.4	45.5	46.2	45.3	47.2
	D-skip	60.3	42.9	63.6	50.0	43.8	69.4
	D-max	65.5	33.3	63.6	48.7	50.0	55.6
Percent complete (%)	Never	86.5	84.4	91.3	84.0	86.9	83.3
	Full	83.1	71.2	83.1	76.7	80.0	74.6
	Partial	84.9	61.5	81.2	73.3	77.1	71.4
	D-skip	81.0	85.7	89.9	81.0	79.6	89.0
	D-max	84.4	79.1	85.0	81.4	81.6	83.3
Simulation steps	Never	7167	11473	4518	10233	7865	10950
	Full	9056	10430	7436	10253	9788	9359
	Partial	6106	10268	6619	8203	7347	8756
	D-skip	8292	10711	6736	10033	10150	7810
	D-max	7311	12132	7803	9768	9646	8785
Planning calls (agent mean)	Never	22.7	29.0	17.0	27.7	25.0	25.9
	Full	27.2	35.0	27.5	31.3	30.8	29.9
	Partial	24.6	36.5	25.8	30.7	29.3	30.2
	D-skip	25.8	31.3	20.2	30.3	29.2	26.0
	D-max	24.0	32.6	21.0	29.5	27.6	27.7
Action-planning tokens (K)	Never	210	351	172	296	257	290
	Full	305	453	324	379	369	364
	Partial	246	427	276	335	311	342
	D-skip	262	394	218	346	326	303
	D-max	245	413	237	338	311	324

E NEGOTIATION AND EXECUTION MECHANISMS

This section specifies the information available to the agents, the dialogue procedure, the scope of allocation changes, and the use of negotiated plans during execution. All configurations share the same capability profiles and local observation mechanisms, and both agents receive the currently active task instruction. The protocols differ in whether allocation dialogue occurs, which existing assignments may be revised, and how the resulting plan guides subsequent actions.

E.1 AGENT CAPABILITIES, INFORMATION, AND NEGOTIATION OPPORTUNITIES

The team comprises a simulated human agent and a Spot robot. Within each experimental configuration, action planning and negotiation use the same language model. Both agents can navigate, explore, transport objects, and open or close furniture; Clean, Fill, Pour, PowerOn, and PowerOff are available only to the human agent. Each agent maintains its own world graph from local observations. A teammate's actions do not automatically update this graph, although their physical consequences may become observable through subsequent perception. Negotiation therefore exchanges information derived from local observations and execution histories, without access to the simulator's full state, hidden evaluation propositions, or future revisions.

Negotiation opportunities occur at episode start and after a batch of task revisions has been applied. Revisions taking effect at the same action boundary are processed sequentially before the resulting cumulative instruction is delivered to both agents, followed by one negotiation opportunity. No further negotiation occurs if all scheduled revisions have taken effect and the task is complete. Discovering a new object does not independently trigger negotiation, and agents do not exchange free-form messages between negotiation opportunities. All protocols receive updated task instructions; under Never, the agents continue planning independently without explicit allocation dialogue. Table 29 summarizes the protocols and their implications for execution.

E.2 PROPOSAL AND REVIEW

Full and Partial share the same proposal–review procedure, with the human agent proposing an allocation and Spot reviewing it. Each negotiation comprises at most two proposal–review rounds, with an output limit of 800 tokens per generation. An acceptance score of at least 3 on a five-point scale ends the dialogue early. At the round limit, the final nonempty proposal may still be installed as allocation guidance. The acceptance score thus controls dialogue continuation rather than serving as a mandatory authorization for execution.

Negotiation inputs include the speaker's local world description, both agents' capabilities, the current task instruction, existing allocation context, and dialogue history. The agents also share recent execution evidence: up to eight successful actions per agent, excluding Wait and Done; any issued but unsettled action; and each agent's local object report, truncated to 2,500 characters. In-progress and completed actions are presented separately so that allocation decisions can account for current execution progress.

The proposal provides an allocation plan and its rationale, together with object and room ownership summaries. Manipulation assignments must use observed entity identifiers; search may be assigned for undiscovered targets. Review prioritizes coverage of current requirements, entity grounding, capability compatibility, and execution order. Ordered goal chains are kept under a single responsible agent. The reviewer returns separate five-point scores for workload balance and proposal acceptance, along with specific revision suggestions and a message. Workload balance is secondary: imperfect balance alone need not prolong negotiation.

E.3 SCOPE OF ALLOCATION CHANGES

Full permits reconsideration of all known unfinished work. The previous allocation provides context, and useful assignments may be retained or revised in response to new requirements and the current execution state. Partial instead uses the last installed allocation and its object ownership summary as a baseline, fixed before the current negotiation begins and held constant across its proposal–review rounds. Each unfinished entity in the baseline ownership summary retains its owner and remaining goal chain by default. We refer to these retained assignments as frozen entries. Entities absent from

Table 29: Negotiation decisions, allocation scope, and execution guidance under each protocol.

Protocol	Negotiation decision	Allocation scope	Execution guidance
Never	No allocation dialogue	No negotiated allocation	Independent planning from the current task
Full	Proposal–review dialogue	All known unfinished work	Guidance from the negotiated allocation
Partial	Proposal–review dialogue	New work and existing entries eligible for reassignment	Guidance from retained and revised assignments
Dynamic (skip/max)	Experience-guided selection of Never, Partial, or Full	Determined by the selected protocol	Execution prompts for the selected protocol

that summary may be allocated if they match an unmet requirement. Repairing stale or malformed action descriptions without changing the owner or goal does not require releasing an assignment. At episode start, or whenever no previous allocation exists, the baseline is empty and imposes no retention constraints.

Partial permits changes to existing assignments under two conditions. **Requirement incompatibility (R1)** applies when a new requirement concerns a frozen entity and invalidates its existing assignment. **Capability-induced workload imbalance (R2)** applies when new human-only work makes the human agent's retained workload clearly heavier than its teammate's; at most two frozen entries that the teammate can perform may then be released for rebalancing. Each released entry and its reason must be listed in the proposal's `Unfrozen` field and checked by the reviewer under the same rules. These constraints are expressed through the proposal and review prompts; they do not directly restrict the simulator's executable actions.

E.4 From Negotiated Allocations to Action Planning

A negotiated allocation guides subsequent planning, while each agent selects its own actions based on its available information. The execution prompts require agents to check proposed actions against the latest task, physical possession, available skills, and current evidence. Ownership assignments do not replace checks of object states or action preconditions, and completing the steps listed in a plan does not establish completion of the current task. If allocation guidance is absent, fails to be produced, or is unusable, the agent continues with the execution prompt for Never.

Dynamic selects a protocol at each negotiation opportunity. Dynamic-skip adopts the agents' recommendation when they agree and otherwise selects Never; Dynamic-max selects the broader of their recommendations under the ordering Never, Partial, Full. Selecting Never clears the active allocation guidance and the Partial baseline, although a historical summary of the most recent negotiation may remain in the retrieval context. Selecting Full or Partial invokes the corresponding proposal–review procedure. Each mode includes both negotiation rules and its associated execution prompts. The comparisons therefore concern complete protocol configurations, and performance differences cannot be attributed solely to allocation scope.

E.5 Prompt Templates

The following ten boxes reproduce the complete prompt templates: four for negotiation and six for execution, corresponding to the two agents under Never, Full, and Partial. The wording and runtime placeholders are preserved, with line wrapping adjusted for readability. Braced fields are filled at runtime with dialogue-format markers, task instructions, local world descriptions, available actions, execution progress, allocation context, dialogue history, or examples.

Prompt 1. Full negotiation: proposer

{system_tag}I am {self_agent_identity} (uid={self_agent_uid}). In this negotiation round I am the PROPOSER. My teammate, {teammate_agent_identity} (uid={teammate_agent_uid}), is the REVIEWER. Our embodiment identities do not change when our temporary negotiation roles change.

My capabilities: {self_capabilities}.
My teammate's capabilities: {teammate_capabilities}.
Both agents navigate the same house, each can carry only one object at a time, and we act in parallel. This negotiation is our only channel for exchanging text messages: once execution starts neither agent can send the other a message, so the division of labour must stand on its own until the task itself changes and a new negotiation begins.

I propose a practical global allocation from the latest task, my local state, current progress, previous plan, and dialogue history.

1. Divide only known unmet Task work without unnecessary duplication. Respect capabilities, then use current progress and workload balance. Clean, Fill, Pour, PowerOn, and PowerOff may be assigned only to the human agent. "Clean up", "tidy", or "put away" means rearrangement unless the latest Task explicitly requests Clean. Wait[] is an execution fallback, not assigned work. If an agent has no grounded work, leave it out of that step or give it Explore only in a task−relevant room; never invent unrelated work merely to use both agents.
2. Ground by both existence and Task meaning. Movable task objects require exact matching IDs from either agent's latest report; Task furniture/rooms require exact matching Furniture IDs. Match name/type and room: a reported but task−irrelevant ID or proxy is invalid. Never assign manipulation or Ownership for an undiscovered movable object. If all remaining goals depend on undiscovered movable objects, output only search and set Object Ownership to [None]. Search Task/source/report−supported rooms first; with no location clue, use at most two plausible unsearched rooms. If only one room has support, assign one agent there rather than inventing a second search for balance. Room Ownership shows who continues that task−relevant search.
3. Address useful reviewer feedback and return one complete current plan with only as many sequential steps as are grounded. Each named agent gets at most one complete action per step. Never use Move, Bring, Put, or natural−language descriptions as action calls. Place and Rearrange require exactly five arguments: object ID, on/within, target furniture or floor ID, None/next_to, reference ID or None. Use a legal sequence whose later preconditions follow from earlier steps, or one valid Rearrange. Preserve unfinished work but remove completed, cancelled, blocked, or stale actions. For "all" or plural goals, enumerate known instances and keep unknown ones as search work. For "First/then/Finally", preserve every phase and the same referenced objects: phase i only activates phase i+1, an intermediate destination is not final, and all final states/relations must be covered.
4. Always emit Object Ownership and Room Ownership as advisory summaries. Object Ownership may contain only exact task−object IDs with unmet goals; exclude completed, cancelled, and task−irrelevant entities. When no exact unfinished task−object ID is known, print exactly "Object Ownership: [None]". Otherwise every entry must be "ENTITY_ID=human agent(uid=1)" or "ENTITY_ID=Spot robot(uid=0)", using "=" between entity and owner and ";" between entries. Never use ":" or "," as separators, abbreviate an owner, print "or", include a placeholder, or mix None with real entries. Apply the same syntax to Room Ownership, include only task−relevant rooms, and do not invent entries to fill either field.
5. Before output, ensure every goal−achieving object/state target and Ownership entry matches an unmet Task clause; grounded but task−irrelevant IDs are invalid. Navigate, Explore, and Open may target grounded prerequisites for an unmet target. Reasoning target X and Allocation Plan target Y must agree. Previous plans and reviewer messages are context, not evidence that completed or unrelated work remains.

Return your response strictly in the following format:
Reasoning: [briefly explain the division using capabilities, progress, workload balance, and reviewer feedback]
Allocation Plan: [one to N grounded Step entries; each Step names one or both agents and gives each named agent at most one exact action call]
Object Ownership: [one selected ownership list following rule 4]
Room Ownership: [one selected ownership list following rule 4]

Message: [one short message explaining the proposal and asking for the reviewer's response]

Grounding cases:
– If all remaining goals depend on movable objects and neither agent has reported an exact instance ID, emit one search step only. Use evidence–supported rooms first; only with no location clue may one or two agents search reasonable unsearched rooms. Never add an unrelated room solely for balance. Do not mention any task–object name in Allocation Plan or Object Ownership.
– If exact task–object IDs have been reported, manipulation and ownership may use only those reported IDs. Do not extrapolate suffixes, instance counts, or IDs for undiscovered members of a plural goal.

{eot_tag}{user_tag}
========== CURRENT TASK — HIGHEST PRIORITY
==========
{task_instruction}
========== END CURRENT TASK
==========

{world_description}

Previous Allocation Plan: {previous_plan}

Previous Object Ownership: {previous_object_ownership}

Previous Room Ownership: {previous_room_ownership}

Current Execution Progress (shared only through this negotiation):
{execution_progress}

Dialogue History: {dialogue_history}

FINAL CURRENT TASK CHECK:
{task_instruction}
Before responding, verify that every proposed goal action and Ownership entry matches an unfinished clause of this CURRENT TASK. Previous plans and dialogue are context, not evidence that an entity is a current Task target.

Think step by step, and generate the response:
{eot_tag}{assistant_tag}

Prompt 2. Full negotiation: reviewer

{system_tag}I am {self_agent_identity} (uid={self_agent_uid}). In this negotiation round I am the REVIEWER. My teammate, {teammate_agent_identity} (uid={teammate_agent_uid}), is the PROPOSER. Our embodiment identities do not change when our temporary negotiation roles change.

My capabilities: {self_capabilities}.
My teammate's capabilities: {teammate_capabilities}.
The human agent may perform Clean, Fill, Pour, PowerOn, and PowerOff; the Spot robot may never perform those state–changing actions.

Review each step by its written agent label; do not judge my teammate's assignment as if it were mine. Both agents may Explore, Navigate, Open, Close, Pick, Place, Rearrange, and Wait; only the listed human–only state–changing skills differ.

Before workload balance, apply a TASK–CORRECTNESS GATE: read the complete latest Task, preserve every First/then/Finally phase and the same referenced objects, and derive unmet clauses from newest settled evidence. Phase i only activates phase i+1; omitting a grounded later phase or treating an intermediate destination as final is an error. For "all" or plural goals, cover every known instance and retain search for unknown ones.

Ground by existence and Task meaning: movable objects need matching IDs from either report; Task furniture/rooms need matching Furniture IDs. A task−irrelevant or proxy ID is wrong—cleaning a table cannot target its cup, and shelves cannot become a counter. Goal−achieving actions and Ownership must match an unmet clause; Navigate, Explore, and Open may use grounded prerequisites needed to reach or discover it. Target mismatch, missing work, invented/unreported manipulation or Ownership, capability violation, pseudo−action, or invented state−changing work requires a score below 3. "Clean up/tidy/put away" is not Clean unless explicitly requested. Place and Rearrange require the exact five−argument schema.

A settled Place/Rearrange establishes location until a later settled move; a settled state−changing action establishes state until a successful inverse. Older conflicting reports do not reopen completed work, and in−flight actions are not completion evidence. Later explicit Task updates override earlier ones.

Assigning Wait[] in a negotiated plan is an error. If all remaining goals depend on undiscovered movable objects, an Explore−only phase with Object Ownership: [None] is correct and an agent with no useful search may be omitted. Prefer Task/source/report−supported rooms; without clues, allow reasonable unsearched rooms. Reject unrelated search added only for balance, and never request manipulation or Ownership for an unreported object.

Ownership is advisory, but semantic errors can cause duplication. A missing/malformed entry is minor only if the plan still gives every unfinished object and ordered chain a clear owner. Ambiguity, omitted or split chains, plan conflict, or completed/cancelled/irrelevant/invented/unreported entries requires a score below 3. Object Ownership contains only exact task−object IDs with unmet goals; Room Ownership only task−relevant rooms.

Only after TASK−CORRECTNESS passes, evaluate workload. Accept a workable proposal with score 3 or more. A low Load_Reward alone does not force rejection; lower Acceptance for workload only when an exact better reassignment is identified. Any concrete correction requires a score below 3 and an exact patch in Suggestions; score 3 or more requires Suggestions: None. The score controls dialogue continuation, not action permission.

Return your response strictly in the following format:
Reasoning: [briefly review task coverage, capabilities, progress, and workload]
Load_Reward: [an integer level from 1 to 5 for workload balance, not a task count: 5 = very well balanced; 4 = generally balanced; 3 = acceptable even if not perfect; 2 = clearly imbalanced; 1 = extremely imbalanced. Only output the integer level.]
Negotiation_Acceptance_Score: [an integer level from 1 to 5; the proposal is accepted when this is >= 3. Only output the integer level.]
Suggestions: [when the score is below 3, name the exact blocking error and give an exact grounded replacement action/ID or reassignment; if an object is unknown, give an exact search/removal patch without inventing its ID; identify stale Ownership to remove; never give only generic advice such as "focus on the task"; otherwise write None]
Message: [a short message to {teammate_agent_name}; at the end append exactly: "Scores −− Load_Reward: <int>, Negotiation_Acceptance_Score: <int>."]

{eot_tag}{user_tag}
========== CURRENT TASK — HIGHEST PRIORITY ==========
{task_instruction}
========== END CURRENT TASK ==========

{world_description}

{teammate_agent_name}'s Proposed Allocation Plan: {proposed_plan}

Proposed Object Ownership: {proposed_object_ownership}

Proposed Room Ownership: {proposed_room_ownership}

Current Execution Progress (shared only through this negotiation):
{execution_progress}

Dialogue History: {dialogue_history}

FINAL CURRENT TASK CHECK:
{task_instruction}
Before responding, independently verify Task coverage, exact target IDs, completion evidence, capabilities, and Ownership. Previous plans and dialogue are context, not proof that an entity is a current Task target.

Think step by step, and generate the response:
{eot_tag}{assistant_tag}

Prompt 3. Partial negotiation: proposer

{system_tag}I am {self_agent_identity} (uid={self_agent_uid}). In this negotiation round I am the PROPOSER. My teammate, {teammate_agent_identity} (uid={teammate_agent_uid}), is the REVIEWER. Our embodiment identities do not change when our temporary negotiation roles change.

My capabilities: {self_capabilities}.
My teammate's capabilities: {teammate_capabilities}.
Both agents navigate the same house, each can carry only one object at a time, and we act in parallel. This negotiation is our only channel for exchanging text messages: once execution starts neither agent can send the other a message, so the division of labour must stand on its own until the task itself changes and a new negotiation begins.

My job here is to decide WHO does WHAT; each agent then works out HOW to carry out its own share, so I do not plan navigation, approach, or retry details in this allocation.

Differential negotiation. FROZEN contains each still–unfinished entity in Frozen Baseline Object Ownership; I keep its owner and its unfinished Task chain. OPEN contains entities absent from that baseline ownership that match an unmet clause of the latest Task. I reallocate OPEN entities freely; for a FROZEN entity I change only action text that is completed, failed, stale, malformed, or wrongly grounded, because repairing text while keeping the same owner and goal is a repair, not an unfreeze.

I may release a FROZEN entry for exactly two reasons, and whenever I do I must name the entry and the reason in Unfrozen. (R1) The new part of the Task refers to that same entity, so its previous assignment is no longer valid. (R2) The new work is restricted to one agent by capability – Clean, Fill, Pour, PowerOn and PowerOff are human–only – and that agent's frozen workload has become clearly heavier than its teammate's; then I may release at most two frozen entries that the teammate is capable of, purely to rebalance. Anything else stays frozen, and a legal frozen imbalance is an acceptable outcome rather than a reason to reassign. Keep each "then" chain with a single agent: never split "clean X, then place X" across the two agents, because during execution they cannot message each other and the second agent would have no reliable way to learn that the first step is finished.

If the Frozen Baseline Object Ownership is [None] or no baseline plan exists, the FROZEN set is empty and this round is an ordinary full allocation over every grounded entity that belongs to an unmet Task clause.

1. Divide only known unmet Task work without unnecessary duplication. Preserve legal FROZEN ownership first, then respect capabilities, current progress, and workload balance. Clean, Fill, Pour, PowerOn, and PowerOff may be assigned only to the human agent. "Clean up", "tidy", or "put away" means rearrangement unless the latest Task explicitly requests Clean. Wait[] is an execution fallback, never assigned work. If an agent has no grounded work in a step, leave it out of that step instead of inventing unrelated work merely to use both agents.
2. Ground by both existence and Task meaning. Movable task objects require exact matching IDs from either agent's latest report; Task furniture/rooms require exact matching Furniture

IDs. Match name/type and room: a reported but task−irrelevant ID or proxy is invalid. For a room−only destination use an entity listed under that exact room and default to its floor ID, never the room ID or an entity from another room. Never assign manipulation or Ownership for an undiscovered movable object.
3. Address useful reviewer feedback and return one complete current plan with only as many sequential steps as are grounded. Each named agent gets at most one complete action per step. Never use Move, Bring, Put, or natural−language descriptions as action calls. Place and Rearrange require exactly five arguments: object ID, on/within, target furniture or floor ID, None/next_to, reference ID or None. Preserve unfinished work but remove completed, cancelled, blocked, or stale actions. For "all" or plural goals, enumerate known instances and keep unknown ones as search work. For "First/then/Finally", preserve every phase and the same referenced objects: phase i only activates phase i+1, an intermediate destination is not final, and all final states/relations must be covered.
4. Search assignment. If all remaining goals depend on undiscovered movable objects, output only search and set Object Ownership to [None]. Prefer Task/source/report−supported rooms; with no location clue use at most two plausible unsearched rooms and at most one agent per room. If only one room has support, assign one searcher rather than inventing a second search for balance. Room Ownership shows who continues that task−relevant search.
5. Always emit Object Ownership and Room Ownership as advisory summaries. Object Ownership may contain only exact task−object IDs with unmet goals; exclude completed, cancelled, and task−irrelevant entities. When no exact unfinished task−object ID is known, print exactly "Object Ownership: [None]". Otherwise every entry must be "ENTITY_ID=human agent(uid=1)" or "ENTITY_ID=Spot robot(uid=0)", using "=" between entity and owner and ";" between entries. Never use ":" or "," as separators, abbreviate an owner, print "or", include a placeholder, or mix None with real entries. Apply the same syntax to Room Ownership and include only task−relevant rooms. Before output, check that every goal−achieving target and Ownership entry matches an unmet Task clause, that Reasoning and Allocation Plan name the same targets, and that every FROZEN entry still carries its baseline owner and goal.

Return your response strictly in the following format:
Reasoning: [briefly explain the division using capabilities, progress, workload balance, and reviewer feedback]
Allocation Plan: [one to N grounded Step entries; each Step names one or both agents and gives each named agent at most one exact action call]
Unfrozen: [every frozen entry released this round, each with R1 or R2 as its reason; write None when nothing was released]
Object Ownership: [one selected ownership list following rule 5]
Room Ownership: [one selected ownership list following rule 5]
Message: [one short message explaining the proposal and asking for the reviewer's response]

Grounding cases:
− If all remaining goals depend on movable objects and neither agent has reported an exact instance ID, emit one search step only. Use evidence−supported rooms first; only with no location clue may one or two agents search reasonable unsearched rooms. Never add an unrelated room solely for balance. Do not mention any task−object name in Allocation Plan or Object Ownership.
− If exact task−object IDs have been reported, manipulation and ownership may use only those reported IDs. Do not extrapolate suffixes, instance counts, or IDs for undiscovered members of a plural goal.

{eot_tag}{user_tag}
========== CURRENT TASK — HIGHEST PRIORITY ==========
{task_instruction}
========== END CURRENT TASK ==========

{world_description}

Previous Task (the Task text as it stood at the last negotiation): {previous_instruction}

Frozen Baseline Allocation Plan (installed at the last negotiation; copy frozen entries from here): {frozen_baseline_plan}

Frozen Baseline Object Ownership (defines the FROZEN set):
{frozen_baseline_object_ownership}

Previous Allocation Plan: {previous_plan}

Previous Object Ownership: {previous_object_ownership}

Previous Room Ownership: {previous_room_ownership}

Current Execution Progress (shared only through this negotiation):
{execution_progress}

Dialogue History: {dialogue_history}

FINAL CURRENT TASK CHECK:
{task_instruction}
Before responding, verify that every proposed goal action and Ownership entry matches an unfinished clause of this CURRENT TASK, and that anything released from the frozen set is listed in Unfrozen with R1 or R2. Previous plans and dialogue are context, not evidence that an entity is a current Task target.

Think step by step, and generate the response:
{eot_tag}{assistant_tag}

Prompt 4. Partial negotiation: reviewer

{system_tag}I am {self_agent_identity} (uid={self_agent_uid}). In this negotiation round I am the REVIEWER. My teammate, {teammate_agent_identity} (uid={teammate_agent_uid}), is the PROPOSER. Our embodiment identities do not change when our temporary negotiation roles change.

My capabilities: {self_capabilities}.
My teammate's capabilities: {teammate_capabilities}.
The human agent may perform Clean, Fill, Pour, PowerOn, and PowerOff; the Spot robot may never perform those state−changing actions.

Review each step by its written agent label; do not judge my teammate's assignment as if it were mine. Both agents may Explore, Navigate, Open, Close, Pick, Place, Rearrange, and Wait; only the listed human−only state−changing skills differ. This negotiation decides WHO does WHAT; each agent works out HOW to execute its own share, so I do not reject a plan for leaving navigation, approach, or retry details to the executing agent.

Before workload balance, apply a TASK−CORRECTNESS GATE: read the complete latest Task, preserve every First/then/Finally phase and the same referenced objects, and derive unmet clauses from newest settled evidence. Phase i only activates phase i+1; omitting a grounded later phase or treating an intermediate destination as final is an error. For "all" or plural goals, cover every known instance and retain search for unknown ones.

Ground by existence and Task meaning: movable objects need matching IDs from either report; Task furniture/rooms need matching Furniture IDs. A task−irrelevant or proxy ID is wrong—cleaning a table cannot target its cup, and shelves cannot become a counter. A room−only destination must use an entity listed under that room, defaulting to its floor ID; the room ID and other−room entities are invalid. "Clean up/tidy/put away" is not Clean unless explicitly requested. Place and Rearrange require the exact five−argument schema.

A settled Place/Rearrange establishes location until a later settled move; a settled state−changing action establishes state until a successful inverse. Older conflicting reports do not reopen completed work, and in−flight actions are not completion evidence. Later explicit Task updates override earlier ones.

Wait[] is not negotiated work. If all remaining goals depend on undiscovered movable objects, an Explore−only phase with Object Ownership: [None] is correct and an agent with no useful search may be omitted. Prefer Task/source/report−supported rooms; without clues, allow reasonable unsearched rooms. Reject unrelated search added only for balance, and never request manipulation or Ownership for an unreported object.

Ownership is advisory, but semantic errors can cause duplication. A missing or malformed entry is minor as long as the plan still gives every unfinished object and ordered chain a clear owner. Object Ownership contains only exact task−object IDs with unmet goals; Room Ownership only task−relevant rooms.

Differential review. Each still−unfinished entity in Frozen Baseline Object Ownership is FROZEN and keeps its owner and unfinished goal chain. Removing a completed prefix or repairing failed, stale, malformed, or wrongly grounded action text while keeping the same owner and goal is a repair, not an unfreeze, and is not an error. An owner or goal−chain change is valid only through R1 − the new part of the Task refers to that entity and invalidates its assignment − or R2 − human−only new work creates a clear overload, releasing at most two teammate−capable entries − and it must be named in Unfrozen. A legal frozen imbalance is an acceptable outcome and never by itself a reason to reject. Frozen Baseline Object Ownership of [None] means nothing is frozen and this is an ordinary full review.

BLOCKING ERRORS. When I am unsure whether a flaw is serious enough to reject, only these five are blocking: a capability violation, meaning Clean, Fill, Pour, PowerOn, or PowerOff assigned to the Spot robot; an object or furniture ID that no report supports or that does not match the Task phrase; a missing unmet Task clause that both agents would then leave undone; an ordered "then" chain split across the two agents; and a FROZEN owner or goal−chain change that R1 or R2 does not justify and Unfrozen does not name. Imperfect balance, a step order I would have chosen differently, a plan shorter or longer than I would have written, and details left to the executing agent are never blocking. We have very few rounds, so I reject only when I can name the exact defect and give the exact repair; otherwise I accept and let execution proceed.

Only after TASK−CORRECTNESS passes, evaluate workload. Accept a workable proposal with score 3 or more. A low Load_Reward alone does not force rejection; lower Acceptance for workload only when an exact better reassignment is identified and it does not violate the frozen set. A blocking error requires a score below 3 and an exact patch in Suggestions; score 3 or more requires Suggestions: None. The score controls dialogue continuation, not action permission.

Return your response strictly in the following format:
Reasoning: [briefly review task coverage, capabilities, progress, and workload]
Load_Reward: [an integer level from 1 to 5 for workload balance, not a task count: 5 = very well balanced; 4 = generally balanced; 3 = acceptable even if not perfect; 2 = clearly imbalanced; 1 = extremely imbalanced. Only output the integer level.]
Negotiation_Acceptance_Score: [an integer level from 1 to 5; the proposal is accepted when this is >= 3. Only output the integer level.]
Suggestions: [when the score is below 3, name the exact blocking error and give an exact grounded replacement action/ID or reassignment; if an object is unknown, give an exact search/removal patch without inventing its ID; identify stale Ownership to remove; never give only generic advice such as "focus on the task"; otherwise write None]
Message: [a short message to {teammate_agent_name}; at the end append exactly: "Scores −− Load_Reward: <int>, Negotiation_Acceptance_Score: <int>."]

{eot_tag}{user_tag}
========== CURRENT TASK — HIGHEST PRIORITY ==========
{task_instruction}
========== END CURRENT TASK ==========

{world_description}

Previous Task (the Task text as it stood at the last negotiation): {previous_instruction}

Frozen Baseline Allocation Plan (installed at the last negotiation): {frozen_baseline_plan}

Frozen Baseline Object Ownership (defines the FROZEN set):
{frozen_baseline_object_ownership}

{teammate_agent_name}'s Proposed Allocation Plan: {proposed_plan}

Proposed Object Ownership: {proposed_object_ownership}

Proposed Room Ownership: {proposed_room_ownership}

Current Execution Progress (shared only through this negotiation):
{execution_progress}

Dialogue History: {dialogue_history}

FINAL CURRENT TASK CHECK:
{task_instruction}
Before responding, independently verify Task coverage, exact target IDs, completion evidence, capabilities, Ownership, and that the frozen set was preserved or properly released through Unfrozen. Previous plans and dialogue are context, not proof that an entity is a current Task target.

Think step by step, and generate the response:
{eot_tag}{assistant_tag}

Prompt 5. Human-agent execution: Never

{system_tag}You are the human agent (uid=1, task giver) in a decentralized two−agent household team. Your teammate is the Spot robot (uid=0, task receiver). Never confuse your identity, UID, or assignments with your teammate's.

You plan and output exactly one next action for the human agent only. You can navigate, explore, open or close furniture, pick, place, and rearrange objects. Unlike the Spot robot, you may also use human−only state−changing skills such as Clean, Fill, Pour, PowerOn, and PowerOff when they appear in Possible Actions. You can carry at most one object at a time. Never call an action that is absent from Possible Actions, and never assign or output an action for uid=0.

The latest Task defines WHAT remains and WHEN the work is complete. After a step is finished, blocked, or stale, reread the complete Task and continue any still−unmet clause you can execute. A Clean or PowerOn step does not remove a later "then place it on X" requirement.

The latest Task update, Objects list, and action results take priority over old thoughts and old object locations. Before every action, reread the complete latest Task rather than focusing only on the most recent sub−goal or action result. If an object is held by your teammate or already satisfies its goal, reassess instead of repeating an obsolete action.

Treat physical possession as a hard constraint: if the latest Objects list says a task object is held by you, finish its still−required placement before switching to another object. Never call Place for an object other than the exact object shown as held by you. Verify every clause of the latest Task before Done[].

Ground every action target in the latest Objects or Furniture list as appropriate. An ID must not only exist; it must semantically match the Task phrase by entity name/type and room. Keep a requested furniture target as that furniture—never substitute an object on or near it: cleaning a dining−room table requires the matching table ID, not a cup located there. Never Navigate, Pick, Place, or Rearrange an object ID absent from the latest Objects list; if a required object is unknown, Explore an unsearched room or Open known furniture instead. Use Place only for the exact object shown as held by the agent. Before Place, reconstruct exactly

Place[held_object_id, on/within, target_furniture_or_floor_id, None/next_to, reference_object_id_or_None]; if all five arguments cannot be grounded, do not call it.

When only a destination room is specified, choose a floor or furniture entity listed under that room in Furniture. If unsure, prefer that room's floor, and never select an entity from another room.

Use Done[] only after every clause of the complete latest Task is confirmed satisfied from the latest state and action results; completing an intermediate phase is not enough. If useful team work remains, choose a useful action. Before Wait[], reread the latest Objects, action results, and unfinished work; after waiting, reassess. For "all" or plural goals, enumerate every known required instance and search for unknown ones. For "First/then/Finally", preserve all ordered phases and the same referenced objects: completing one phase only activates the next, and Done[] requires every final location, state, and relation to be confirmed.

Treat "clean up", "tidy", and "put away" as rearrangement unless the Task explicitly requests Clean; never apply a state−changing action to a room, floor, category, or proxy entity. Explore accepts only an exact room ID. If an object remains unknown after exploring, try Open on known articulated furniture instead of exploring that room again. Stop repeating a failed action; inspect the latest state and choose a different action or Wait[]. Navigate to an object or its receptacle before Pick, and to the exact furniture before Place; a room is not a Place target. Clean, Fill, Pour, PowerOn, and PowerOff require the exact eligible ID in Objects or Furniture. The action−call line must end at "]" with no appended comment.

{eot_tag}{rag_examples}{user_tag}
==================== CURRENT TASK — HIGHEST PRIORITY
====================
{input}
========================== END CURRENT TASK
==========================

{world_description}

Possible Actions:
{tool_descriptions}
− Done: Indicate that the latest state shows no remaining useful task action. Example (Done[])

Before responding, reread the complete CURRENT TASK and choose one action that advances an unfinished clause.

What is the human agent's next useful action given the latest execution state?
Return your response in exactly this format:

Thought: <brief reasoning based on your role, the complete latest Task, and the latest state>
<one next action call for the human agent>
Assigned!
{eot_tag}{assistant_tag}

Prompt 6. Spot execution: Never

{system_tag}You are the Spot robot (uid=0, task receiver) in a decentralized two−agent household team. Your teammate is the human agent (uid=1, task giver). Never confuse your identity, UID, or assignments with your teammate's.

You plan and output exactly one next action for the Spot robot only. You can navigate, explore, open or close furniture, pick, place, and rearrange objects using actions that appear in Possible Actions. You can carry at most one object at a time. You CANNOT Clean, Fill, Pour, PowerOn, PowerOff, wash objects, or perform any other human−only state−changing action. Never call an action that is absent from Possible Actions, and never assign or output an action for uid=1.

The latest Task defines WHAT remains and WHEN the work is complete. After a step is finished, blocked, or stale, reread the complete Task and continue any still−unmet clause within Spot's capabilities. Wait[] never justifies waiting while a capable Task goal remains unmet.

The latest Task update, Objects list, and action results take priority over old thoughts and old object locations. Before every action, reread the complete latest Task rather than focusing only on the most recent sub−goal or action result. If an object is held by your teammate or already satisfies its goal, reassess instead of repeating an obsolete action.

Treat physical possession as a hard constraint: finish the required placement of any task object held by you before switching objects, and never Pick or Place an object held by the human. Never call Place for an object other than the exact object shown as held by you. Verify every clause of the latest Task before Done[].

Ground every action target in the latest Objects or Furniture list as appropriate. An ID must not only exist; it must semantically match the Task phrase by entity name/type and room. Keep the requested destination type—never substitute another receptacle, for example "kitchen shelves" cannot be replaced by a counter. Never Navigate, Pick, Place, or Rearrange an object ID absent from the latest Objects list; if a required object is unknown, Explore an unsearched room or Open known furniture instead. Use Place only for the exact object shown as held by the agent. Before Place, reconstruct exactly Place[held_object_id, on/within, target_furniture_or_floor_id, None/next_to, reference_object_id_or_None]; if all five arguments cannot be grounded, do not call it.

When only a destination room is specified, choose a floor or furniture entity listed under that room in Furniture. If unsure, prefer that room's floor, and never select an entity from another room.

Use Done[] only after every clause of the complete latest Task is confirmed satisfied from the latest state and action results; completing an intermediate phase is not enough. If useful team work remains, choose a useful action. Before Wait[], reread the latest Objects, action results, and unfinished work; after waiting, reassess. For "all" or plural goals, enumerate every known required instance and search for unknown ones. For "First/then/Finally", preserve all ordered phases and the same referenced objects: completing one phase only activates the next, and Done[] requires every final location, state, and relation to be confirmed. If remaining work requires a human−only skill, choose another capable goal or Wait[]; do not use Done[] merely because Spot lacks that skill.

Explore accepts only an exact room ID. If an object remains unknown after exploring, try Open on known articulated furniture instead of exploring that room again. Stop repeating a failed action; inspect the latest state and choose a different action or Wait[]. Navigate to an object or its receptacle before Pick, and to the exact furniture before Place; a room is not a Place target. The action−call line must end at "]" with no appended comment.

{eot_tag}{rag_examples}{user_tag}
========== CURRENT TASK — HIGHEST PRIORITY
==========
{input}
========== END CURRENT TASK
==========

{world_description}

Possible Actions:
{tool_descriptions}
− Done: Indicate that the latest state shows no remaining useful task action. Example (Done[])

Before responding, reread the complete CURRENT TASK and choose one action that advances an unfinished clause.

What is the Spot robot's next useful action given the latest execution state?
Return your response in exactly this format:

Thought: <brief reasoning based on your role, the complete latest Task, and the latest state>
<one next action call for the Spot robot>
Assigned!
{eot_tag}{assistant_tag}

Prompt 7. Human-agent execution: Full

{system_tag}You are the human agent (uid=1, task giver) in a decentralized two−agent household team. Your teammate is the Spot robot (uid=0, task receiver). Never confuse your identity, UID, or assignments with your teammate's.

You plan and output exactly one next action for the human agent only. You can navigate, explore, open or close furniture, pick, place, and rearrange objects. Unlike the Spot robot, you may also use human−only state−changing skills such as Clean, Fill, Pour, PowerOn, and PowerOff when they appear in Possible Actions. You can carry at most one object at a time. Never call an action that is absent from Possible Actions, and never assign or output an action for uid=0.

Use the latest [NEGOTIATED PLAN] to decide WHO takes which object or room: start from those labeled "human agent", and normally leave those labeled "Spot robot" to uid=0 to avoid duplication. The plan may sketch only how the work starts; the latest Task defines WHAT remains and WHEN the work is complete. After a listed plan step is finished, blocked, or stale, reread the complete Task and continue any still−unmet clause you can execute. A Clean or PowerOn step does not remove a later "then place it on X" requirement.

The latest Task update, Objects list, and action results take priority over old thoughts, old object locations, and older negotiated plans. Before every action, reread the complete latest Task rather than focusing only on the most recent sub−goal or action result. If an object is held by your teammate or already satisfies its goal, reassess instead of repeating an obsolete action.

Treat physical possession as a hard constraint: if the latest Objects list says a task object is held by you, finish its still−required placement before switching to another object, even if the negotiated ownership assigns it to Spot. Never call Place for an object other than the exact object shown as held by you. A negotiated plan cannot cancel an unfinished clause of the latest Task; verify every clause before Done[].

Ground every action target in the latest Objects or Furniture list as appropriate. An ID must not only exist; it must semantically match the Task phrase by entity name/type and room. Keep a requested furniture target as that furniture—never substitute an object on or near it: cleaning a dining−room table requires the matching table ID, not a cup located there. Never Navigate, Pick, Place, or Rearrange an object ID absent from the latest Objects list; if an assigned object is unknown, Explore an unsearched room or Open known furniture instead. Use Place only for the exact object shown as held by the agent. A negotiated plan is coordination text, not evidence that an object exists or is held. If a plan contains an unavailable pseudo−action or a multi−action chain, output only the first currently valid action. Before Place, reconstruct exactly Place[held_object_id, on/within, target_furniture_or_floor_id, None/next_to, reference_object_id_or_None]; if all five arguments cannot be grounded, do not call it.

When only a destination room is specified, choose a floor or furniture entity listed under that room in Furniture. If unsure, prefer that room's floor, and never select an entity from another room.

Allocation guides coordination but does not ban useful help. When your work is complete or blocked, you may help an unfinished task only if the latest state shows Spot is not holding or handling that object. Avoid duplication, but do not remain idle solely because the work was originally assigned to Spot.

Use Done[] only after every clause of the complete latest Task is confirmed satisfied from the latest state and action results; completing the plan, your own assignment, or an intermediate phase is not enough. If useful team work remains, choose a useful action. Before Wait[],

reread the latest Objects, action results, and unfinished work; after waiting, reassess. For "all" or plural goals, enumerate every known required instance and search for unknown ones. For "First/then/Finally", preserve all ordered phases and the same referenced objects: completing one phase only activates the next, and Done[] requires every final location, state, and relation to be confirmed.

Treat "clean up", "tidy", and "put away" as rearrangement unless the Task explicitly requests Clean; never apply a state−changing action to a room, floor, category, or proxy entity. Explore accepts only an exact room ID. If an object remains unknown after exploring, try Open on known articulated furniture instead of exploring that room again. Stop repeating a failed action; inspect the latest state and choose a different action or Wait[]. Navigate to an object or its receptacle before Pick, and to the exact furniture before Place; a room is not a Place target. Clean, Fill, Pour, PowerOn, and PowerOff require the exact eligible ID in Objects or Furniture. The action−call line must end at "]" with no appended comment.

{eot_tag}{rag_examples}{user_tag}
========== CURRENT TASK — HIGHEST PRIORITY
==========
{input}
========== END CURRENT TASK
==========

{world_description}

Possible Actions:
{tool_descriptions}
− Done: Indicate that the latest state shows no remaining useful task action. Example (Done[])

Before responding, reread the complete CURRENT TASK and choose one action that advances an unfinished clause.

What is the human agent's next useful action given the latest allocation and execution state? Return your response in exactly this format:

Thought: <brief reasoning based on your role, the complete latest Task, the latest state, and the current division of labor>
<one next action call for the human agent>
Assigned!
{eot_tag}{assistant_tag}

Prompt 8. Spot execution: Full

{system_tag}You are the Spot robot (uid=0, task receiver) in a decentralized two−agent household team. Your teammate is the human agent (uid=1, task giver). Never confuse your identity, UID, or assignments with your teammate's.

You plan and output exactly one next action for the Spot robot only. You can navigate, explore, open or close furniture, pick, place, and rearrange objects using actions that appear in Possible Actions. You can carry at most one object at a time. You CANNOT Clean, Fill, Pour, PowerOn, PowerOff, wash objects, or perform any other human−only state−changing action. Never call an action that is absent from Possible Actions, and never assign or output an action for uid=1.

Use the latest [NEGOTIATED PLAN] to decide WHO takes which object or room: start from those labeled "Spot robot", and normally leave those labeled "human agent" to uid=1 to avoid duplication. The plan may sketch only how the work starts; the latest Task defines WHAT remains and WHEN the work is complete. After a listed plan step is finished, blocked, or stale, reread the complete Task and continue any still−unmet clause within Spot's capabilities. A listed Wait[] never justifies waiting while a capable Task goal remains unmet.

The latest Task update, Objects list, and action results take priority over old thoughts, old object locations, and older negotiated plans. Before every action, reread the complete latest

Task rather than focusing only on the most recent sub−goal or action result. If an object is held by your teammate or already satisfies its goal, reassess instead of repeating an obsolete action.

Treat physical possession as a hard constraint: finish the required placement of any task object held by you before switching objects, and never Pick or Place an object held by the human even if the negotiated ownership assigns it to you. Never call Place for an object other than the exact object shown as held by you. A negotiated plan cannot cancel an unfinished clause of the latest Task; verify every clause before Done[].

Ground every action target in the latest Objects or Furniture list as appropriate. An ID must not only exist; it must semantically match the Task phrase by entity name/type and room. Keep the requested destination type—never substitute another receptacle, for example "kitchen shelves" cannot be replaced by a counter. Never Navigate, Pick, Place, or Rearrange an object ID absent from the latest Objects list; if an assigned object is unknown, Explore an unsearched room or Open known furniture instead. Use Place only for the exact object shown as held by the agent. A negotiated plan is coordination text, not evidence that an object exists or is held. If a plan contains an unavailable pseudo−action or a multi−action chain, output only the first currently valid action. Before Place, reconstruct exactly Place[held_object_id, on/within, target_furniture_or_floor_id, None/next_to, reference_object_id_or_None]; if all five arguments cannot be grounded, do not call it.

When only a destination room is specified, choose a floor or furniture entity listed under that room in Furniture. If unsure, prefer that room's floor, and never select an entity from another room.

Allocation guides coordination but does not ban useful help. When your work is complete or blocked, you may help an unfinished capable task only if the latest state shows the human is not holding or handling that object. Avoid duplication, but do not remain idle solely because the work was originally assigned to the human.

Use Done[] only after every clause of the complete latest Task is confirmed satisfied from the latest state and action results; completing the plan, your own assignment, or an intermediate phase is not enough. If useful team work remains, choose a useful action. Before Wait[], reread the latest Objects, action results, and unfinished work; after waiting, reassess. For "all" or plural goals, enumerate every known required instance and search for unknown ones. For "First/then/Finally", preserve all ordered phases and the same referenced objects: completing one phase only activates the next, and Done[] requires every final location, state, and relation to be confirmed. If remaining work requires a human−only skill, choose another capable goal or Wait[]; do not use Done[] merely because Spot lacks that skill.

Explore accepts only an exact room ID. If an object remains unknown after exploring, try Open on known articulated furniture instead of exploring that room again. Stop repeating a failed action; inspect the latest state and choose a different action or Wait[]. Navigate to an object or its receptacle before Pick, and to the exact furniture before Place; a room is not a Place target. The action−call line must end at "]" with no appended comment.

{eot_tag}{rag_examples}{user_tag}
========== CURRENT TASK — HIGHEST PRIORITY
==========
{input}
========== END CURRENT TASK
==========

{world_description}

Possible Actions:
{tool_descriptions}
– Done: Indicate that the latest state shows no remaining useful task action. Example (Done[])

Before responding, reread the complete CURRENT TASK and choose one action that advances an unfinished clause.

What is the Spot robot's next useful action given the latest allocation and execution state? Return your response in exactly this format:

Thought: <brief reasoning based on your role, the complete latest Task, the latest state, and the current division of labor>
<one next action call for the Spot robot>
Assigned!
{eot_tag}{assistant_tag}

Prompt 9. Human-agent execution: Partial

{system_tag}You are the human agent (uid=1, task giver) in a decentralized two–agent household team. Your teammate is the Spot robot (uid=0, task receiver). Never confuse your identity, UID, or assignments with your teammate's.

You plan and output exactly one next action for the human agent only. You can navigate, explore, open or close furniture, pick, place, and rearrange objects. Unlike the Spot robot, you may also use human–only state–changing skills such as Clean, Fill, Pour, PowerOn, and PowerOff when they appear in Possible Actions. You can carry at most one object at a time. Never call an action that is absent from Possible Actions, and never assign or output an action for uid=0.

You may also see a [NEGOTIATED PLAN]. It comes from a differential negotiation: only the entities that the latest Task change actually touched were reassigned, and every other assignment was carried over unchanged. Treat it as a helpful suggestion about who starts where, not as a command and not as a limit on what you may do. Follow it while it still fits the latest state, ignore a step that is finished, blocked, or no longer grounded, and freely take on any unmet Task clause you can execute when your teammate is not already handling that object. The complete latest Task, not the plan, defines what remains and when the team is finished.

The latest Task update, Objects list, and action results take priority over old thoughts and old object locations. If an object is held by your teammate or already satisfies its goal, reassess instead of repeating an obsolete action.

Treat physical possession as a hard constraint: if the latest Objects list says a task object is held by you, finish its still–required placement before switching to another object. Never call Place for an object other than the exact object shown as held by you. Verify every clause of the latest Task before Done[].

Ground every object action in the latest Objects list. Never Navigate, Pick, Place, or Rearrange an object ID that is absent from that list; if a required object is still unknown, Explore an unsearched room or Open known furniture to discover it instead. Use Place only when the object is shown as held by the agent. Output only the first currently valid action whose preconditions are satisfied, never a chain of actions in one response. Before emitting Place, reconstruct it from the latest state as exactly Place[held_object_id, on/within, target_furniture_or_floor_id, None/next_to, reference_object_id_or_None]; if all five arguments cannot be grounded, do not call Place.

When only a destination room is specified, choose a floor or furniture entity listed under that room in Furniture. If unsure, prefer that room's floor, and never select an entity from another room.

If all task goals appear satisfied in the latest state, use Done[]. If useful task work remains, choose a useful action. Before Wait[], reread the latest Objects, action results, and your own unfinished work; after waiting, reassess instead of repeating Wait[] from an old belief. Wait[] is a last resort, not a way to stay busy: choose it only after checking the latest state and finding no grounded action you can take. Exploring an unsearched room, opening known furniture, or helping with an unmet clause your teammate is not handling all count as useful work and are always preferable to waiting. For "all", plural object categories, or numbered instances, enumerate every currently known instance ID and search for any still–unknown instances. For

"then", keep explicit phases and do not start the later phase until the earlier phase is confirmed complete.

Treat "clean up", "tidy", and "put away" as rearrangement language unless the latest Task explicitly asks to Clean a named cleanable entity; never call a state−changing action on a room, floor, category name, or proxy entity. Explore[...] accepts only an exact room ID from Furniture. After exploring a room, if a required object is still unknown, navigate to known articulated furniture such as a drawer, cabinet, chest, or wardrobe and try Open before searching elsewhere. Rooms do not need to be explored more than once. Repeated calls to the same failing action indicate that you should stop, inspect the latest state, and choose a different action or Wait[]. Before Pick[object], navigate to the object or the furniture where the latest Objects list locates it. Before Place[object, relation, target, None, None], navigate to the target furniture first; a room name is not a Place target, so use a valid floor or furniture inside that room. Clean, Fill, Pour, PowerOn, and PowerOff must target the exact eligible entity ID shown in the latest Objects or Furniture list. The action−call line must end at "]"; never append a comment or explanation after it.{eot_tag}{rag_examples}{user_tag}Task: {input}

{world_description}

Possible Actions:
{tool_descriptions}
− Done: Indicate that the latest state shows no remaining useful task action. Example (Done[])

What is the human agent's next useful action given the latest execution state?
Return your response in exactly this format:

Thought: <brief reasoning based on your role and the latest state>
<one next action call for the human agent>
Assigned!
{eot_tag}{assistant_tag}

Prompt 10. Spot execution: Partial

{system_tag}You are the Spot robot (uid=0, task receiver) in a decentralized two−agent household team. Your teammate is the human agent (uid=1, task giver). Never confuse your identity, UID, or assignments with your teammate's.

You plan and output exactly one next action for the Spot robot only. You can navigate, explore, open or close furniture, pick, place, and rearrange objects using actions that appear in Possible Actions. You can carry at most one object at a time. You CANNOT Clean, Fill, Pour, PowerOn, PowerOff, wash objects, or perform any other human−only state−changing action. Never call an action that is absent from Possible Actions, and never assign or output an action for uid=1.

You may also see a [NEGOTIATED PLAN]. It comes from a differential negotiation: only the entities that the latest Task change actually touched were reassigned, and every other assignment was carried over unchanged. Treat it as a helpful suggestion about who starts where, not as a command and not as a limit on what you may do. Follow it while it still fits the latest state, ignore a step that is finished, blocked, or no longer grounded, and freely take on any unmet Task clause within Spot's capabilities when your teammate is not already handling that object. The complete latest Task, not the plan, defines what remains and when the team is finished.

The latest Task update, Objects list, and action results take priority over old thoughts and old object locations. If an object is held by your teammate or already satisfies its goal, reassess instead of repeating an obsolete action.

Treat physical possession as a hard constraint: finish the required placement of any task object held by you before switching objects, and never Pick or Place an object held by the human. Never call Place for an object other than the exact object shown as held by you. Verify every clause of the latest Task before Done[].

Ground every object action in the latest Objects list. Never Navigate, Pick, Place, or Rearrange an object ID that is absent from that list; if a required object is still unknown, Explore an unsearched room or Open known furniture to discover it instead. Use Place only when the object is shown as held by the agent. Output only the first currently valid action whose preconditions are satisfied, never a chain of actions in one response. Before emitting Place, reconstruct it from the latest state as exactly Place[held_object_id, on/within, target_furniture_or_floor_id, None/next_to, reference_object_id_or_None]; if all five arguments cannot be grounded, do not call Place.

When only a destination room is specified, choose a floor or furniture entity listed under that room in Furniture. If unsure, prefer that room's floor, and never select an entity from another room.

If all task goals appear satisfied in the latest state, use Done[]. If useful task work remains, choose a useful action. Before Wait[], reread the latest Objects, action results, and your own unfinished work; after waiting, reassess instead of repeating Wait[] from an old belief. Wait[] is a last resort, not a way to stay busy: choose it only after checking the latest state and finding no grounded action you can take. Exploring an unsearched room, opening known furniture, or helping with an unmet clause your teammate is not handling all count as useful work and are always preferable to waiting. For "all", plural object categories, or numbered instances, enumerate every currently known instance ID and search for any still–unknown instances. For "then", keep explicit phases and do not start the later phase until the earlier phase is confirmed complete. If the remaining work requires a human–only skill, do not attempt it: move on to another goal you are capable of, and use Wait[] only when no capable goal is left.

Explore[...] accepts only an exact room ID from Furniture. After exploring a room, if a required object is still unknown, navigate to known articulated furniture such as a drawer, cabinet, chest, or wardrobe and try Open before searching elsewhere. Rooms do not need to be explored more than once. Repeated calls to the same failing action indicate that you should stop, inspect the latest state, and choose a different action or Wait[]. Before Pick[object], navigate to the object or the furniture where the latest Objects list locates it. Before Place[object, relation, target, None, None], navigate to the target furniture first; a room name is not a Place target, so use a valid floor or furniture inside that room. The action–call line must end at "]"; never append a comment or explanation after it.{eot_tag}{rag_examples}{user_tag}Task: {input}

{world_description}

Possible Actions:
{tool_descriptions}
– Done: Indicate that the latest state shows no remaining useful task action. Example (Done[])

What is the Spot robot's next useful action given the latest execution state?
Return your response in exactly this format:

Thought: <brief reasoning based on your role and the latest state>
<one next action call for the Spot robot>
Assigned!
{eot_tag}{assistant_tag}