CCL: COUNTERFACTUAL CONTRACT LEARNING FOR LOCAL EXECUTION RECOVERY IN EMBODIED MULTI-AGENT COLLABORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In embodied multi-agent collaboration, agents must coordinate their actions since completing one action may depend on unfinished work by another agent. Existing approaches use execution feedback to revise plans and generate recovery actions. However, resolving the immediate failure does not ensure completion of the affected subtask. Such corrections require verification before they can serve as successful training examples. We propose Counterfactual Contract Learning (CCL), which learns coordinated repair programs from alternatives verified against a contract specifying the local goal and previously satisfied conditions to preserve. Specifically, we initialize a repair model through supervised fine-tuning on verified programs. The model then generates a direct repair and samples alternatives from the same input. By replaying these programs from the same saved state under the fixed contract, we identify successful alternatives to failed direct repairs as training targets. During training, CCL aligns the paired programs and emphasizes the first differing action, while weaker supervision on subsequent actions teaches complete repairs. For evaluation, we construct PARTNR-FS, a benchmark of recorded failure states with verified repair examples. CCL improves strict recovery rates over Repair-SFT across four language models and, when integrated into CoELA and PCE, improves final object transport and transport progress on TDW-MAT.

1 INTRODUCTION

In embodied multi-agent collaboration, an action may fail because another agent has not completed a prerequisite. In Figure 1(A), Alice holds a toy airplane and Bob a helmet. Their goal is to place both objects next to each other on a bedroom shelf. Placing only the airplane frees one hand but leaves the shared goal incomplete until the helmet is placed beside it. Such dependencies also arise in PARTNR (Chang et al., 2025), where progress toward a shared goal can depend on how agents coordinate their actions. Recovery must therefore extend beyond making an individual action executable to completing the affected goal through coordinated actions (Zhang et al., 2024; Liu et al., 2025; Zhang et al., 2025).

We study this distinction through *local execution recovery*, where a model learns short programs of skill calls that complete the affected goal while preserving completed work. Execution feedback supports replanning and staged recovery from the current failure (Singh et al., 2022; Farag et al., 2025), and learning from these corrections can improve future attempts (Song et al., 2024; Wang et al., 2024). However, using a proposed repair as supervision requires verifying that it completes the original goal from the recorded failure state. Even when a repair succeeds, the outcome of the complete program does not indicate which decisions should receive greater emphasis during training. A challenge also examined in process supervision and reasoning interventions (Lightman et al., 2024; Yang et al., 2026). Learning local recovery therefore requires both verified repair targets and a way to allocate supervision within them.

These two requirements motivate **Counterfactual Contract Learning (CCL)**, which uses replay to verify alternatives generated by the repair model itself and program alignment to guide supervision, as illustrated in Figure 1(B). We first initialize the model through supervised fine-tuning on verified repair examples (Repair-SFT) and then hold it fixed during data collection. For each failure state,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

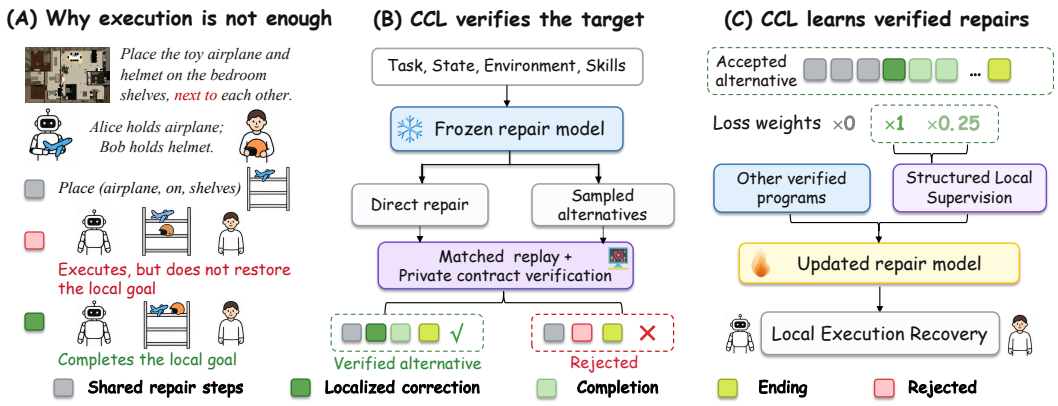


Figure 1: **Learning repairs that complete the local goal.** (A) Executable actions can leave the shared goal incomplete. (B) Replay selects verified alternatives to a direct repair. (C) Primary loss weights emphasize the first changed action. Shared steps are identical initial actions in both programs.

the frozen model generates a direct repair without sampling and samples alternatives from the same input, providing candidate programs whose outcomes and action choices can be compared.

To compare their outcomes, we replay these proposals from the same saved execution state and task history against a fixed *recovery contract* specifying the goal and conditions to preserve. An alternative qualifies when it succeeds in every designated replay and the direct repair fails at least one. We also require matching recorded states after shared initial calls to ensure that the differing decisions begin under common recorded conditions. The selected alternative undergoes manual semantic review before becoming a training target. The contract guides acceptance without entering the model input.

Once an alternative has been verified, its alignment with the direct repair guides the allocation of supervision, as illustrated in Figure 1(C). Because the shared initial calls are identical in both programs, they remain prediction context without receiving direct correction loss. Training instead emphasizes the first changed action, including its agent, skill, and argument values. While this emphasis directs learning toward the earliest change, lower-weight supervision on subsequent actions and the ending teaches the model how to complete the repair. Alongside this correction objective, verified repairs from other states provide a separate preservation objective to reinforce previously successful behavior. After training, the updated model generates each repair in one call without simulator search.

To evaluate local execution recovery, we construct **PARTNR-FS** from recorded failure states with saved checkpoints and verified repairs. These enable comparisons from the same states, where CCL improves strict recovery rates across models and repair methods. We then test whether the local gains benefit complete tasks by adding CCL to CoELA and PCE on TDW-MAT. Both controllers improve final object transport and progress.

Our contributions are as follows: (1) **A learning problem for local recovery.** We formulate repair as learning short programs that complete the affected task goal while respecting dependencies between embodied agents. (2) **Counterfactual Contract Learning.** We connect verification of complete repair programs with supervision on changed decisions, while retaining training on the continuation. (3) **PARTNR-FS benchmark construction.** We construct a benchmark of recorded failure states with verified repair examples, enabling alternatives to be compared from the same execution state. (4) **Recovery and cooperative task improvements.** CCL improves overall recovery over Repair-SFT across language models and leads the repair-method comparison on PARTNR-FS. Adding CCL to CoELA and PCE also improves cooperative transport on TDW-MAT.

2 Related Work

Embodied planning. Language-based planners connect instructions to available robot skills and observed environments (Ahn et al., 2022; Song et al., 2023; Singh et al., 2022), with generated pro-

grams making action sequences explicit (Singh et al., 2022). When execution fails, Inner Monologue uses the feedback to revise planning (Huang et al., 2022). REFLECT summarizes such experiences to guide correction, while CMFR structures the recovery process into stages (Liu et al., 2023; Farag et al., 2025). In collaborative tasks, such corrections must also account for partner progress and the spatial or temporal dependencies represented in PARTNR (Chang et al., 2025). To manage these dependencies, CoELA combines communication with planning, CaPo adapts cooperative plans, and COMBO predicts joint action effects (Zhang et al., 2024; Liu et al., 2025; Zhang et al., 2025). As the execution context changes, REVECA adapts decisions to new information and PCE accounts for planning uncertainty (Seo et al., 2025; 2026).

Learning from agent experience. Beyond revising the current plan, execution experience can improve later decisions. Reflexion retains verbal feedback in memory (Shinn et al., 2023), whereas ETO and E^{2}CL use exploratory trajectories to train on preferences or corrections (Song et al., 2024; Wang et al., 2024). AgentRefine and Agent-R likewise train on revised trajectories (Fu et al., 2025; Yuan et al., 2025), while LLawCo learns behavioral rules from failures in embodied cooperation (Zhou et al., 2026). Other approaches change the learning signal applied to generated behavior: SCoRe uses reinforcement learning for self-correction, whereas GKD supplies teacher token probabilities on student outputs (Kumar et al., 2025; Agarwal et al., 2024). Rather than correcting behavior under its original instruction, HER and HSL relabel goals to match achieved outcomes, turning that behavior into successful examples (Andrychowicz et al., 2017; Li et al., 2026b).

Credit assignment. Once experience has supplied training examples, a further question is which decisions within them to reinforce. Process supervision evaluates intermediate steps rather than only final answers (Lightman et al., 2024; Setlur et al., 2025). IPR and STeCa use stepwise refinement, while HPL combines preferences across trajectory levels (Xiong et al., 2024; Wang et al., 2025; Gao et al., 2026). Alternative decisions offer another basis for supervision: InT trains on proposed reasoning corrections and their preceding steps before reinforcement learning (Yang et al., 2026), while CSO verifies expert replacements through policy continuations (Li et al., 2026a). Reexecution also tests failure hypotheses in DoVer and CausalFlow by measuring the effects of changes to messages, plans, or steps (Ma et al., 2026; Bonagiri et al., 2026).

Our work. CCL brings verified alternatives and localized supervision to embodied repair while keeping the original local goal fixed. Replay and semantic review select complete repairs, which are then aligned with the direct repair to locate changed decisions. This alignment focuses correction training, while training on other verified successes supports retention. PARTNR-FS evaluates the resulting local repairs, and TDW-MAT tests their contribution within cooperative controllers.

3 Counterfactual Contract Learning

CCL first trains a repair model on verified examples, then holds it fixed while collecting direct repairs and sampled alternatives. Verification selects successful alternatives as training targets, and alignment identifies their changes for focused correction alongside training on independent successes.

3.1 Bounded Recovery in Multi-Agent Execution

We study a multi-agent setting in which embodied agents pursue a shared instruction. Recovery may require assigning a prerequisite to one agent before another can complete the local goal, as illustrated by Alice and Bob in Figure 1(A). The repair must therefore coordinate both actor choice and action order. At a failure state, a single repair model receives public input x: the instruction, observed agent and object states, recent history, the failed action and feedback, and available skills. Conditioned on x, the model with parameters θ generates $y = (a_1, \ldots, a_T)$ according to $\pi_\theta(y \mid x)$. Each call $a_j = (i_j, k_j, p_j)$ specifies the executing agent i_j, skill k_j, and argument list p_j at position j. The agent identifier and skill must be available in the public action interface. The constraint $T \leq 5$ applies to the whole repair program and counts skill calls rather than simulator steps.

To determine whether these calls repair the failure, we associate x with a private record c containing the simulator checkpoint and task history. A recovery contract $\mathcal{C}$ specifies the local goal and previously satisfied conditions, both fixed before candidate testing. Given these records, the binary verifier

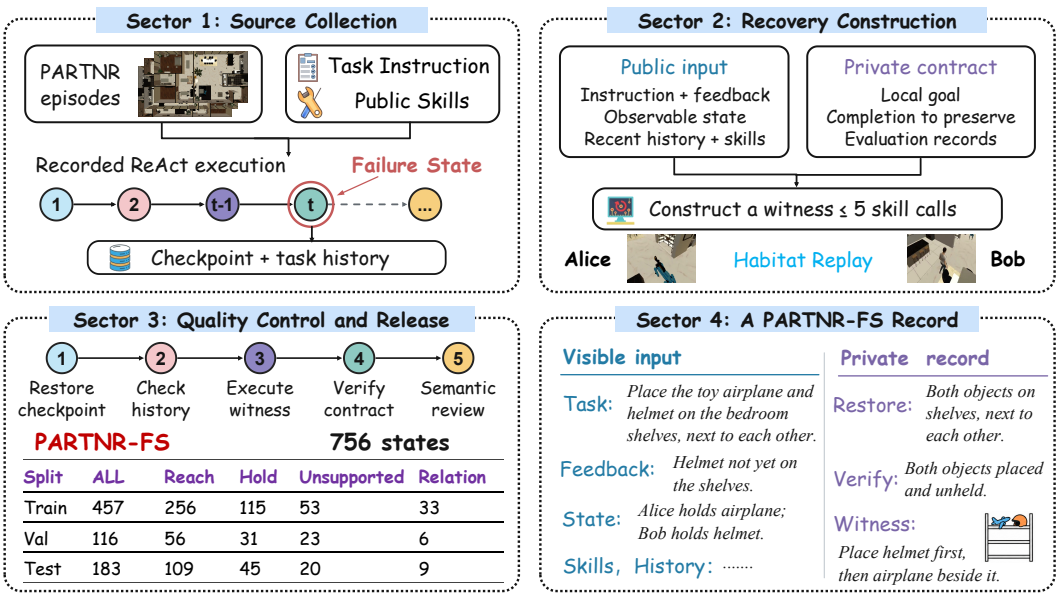


Figure 2: **Construction of PARTNR-FS.** Each of the 756 failure states contains public model inputs and separate verification records. The illustrated witness comes from training episode 1425 and supplies a verified repair for dataset construction.

$V(c, y, z; \mathcal{C})$ evaluates program y under simulator seed z. It returns one when restoration is correct and execution completes the local goal, preserves the protected conditions, and passes checks for repeated failure or loss of completed work during the repair; otherwise it returns zero. Thus x conditions generation, while c and $\mathcal{C}$ determine acceptance. Appendices C and D give the interface and checks. Table A.1 collects the notation used throughout the paper.

3.2 Constructing PARTNR-FS

To obtain failure states with these public inputs and private records, we record local events during cooperative PARTNR episodes controlled by GPT-4o with ReAct (Chang et al., 2025; Yao et al., 2022; Hurst et al., 2024). For each event, we save the agent and object states supplied by Habitat (Savva et al., 2019; Puig et al., 2024), together with its task history and recovery contract, even if the original episode later recovers. We construct each recovery contract by automatically linking the recorded failure context to official task predicates. We admit a state to PARTNR-FS only after analysis or programmatic search finds a short repair that passes execution checks and semantic review. This *recovery witness* establishes that the failure can be resolved within the call budget and, for training states, supplies the Repair-SFT target. The witness in Figure 2, for example, places the helmet first so that the other agent can place the airplane beside it.

The resulting dataset contains 457 training, 116 validation, and 183 test states across reachability, holding, unsupported actions, and relation failures. The full construction and validation process involved more than 30,000 simulator replays. Training witnesses initialize the model, while the saved states support a comparison of its later proposals. Appendix B details this construction.

3.3 Repair Model Initialization

Repair-SFT establishes the output interface and repair behavior on which CCL builds. Each training input is paired with a verified witness serialized as JSON agent, skill, and argument fields. We minimize cross entropy (CE) over the full response, including the native ending token, while masking prompt tokens. This teaches complete feasible programs before CCL distinguishes shared from changed actions; private contracts remain outside the input.

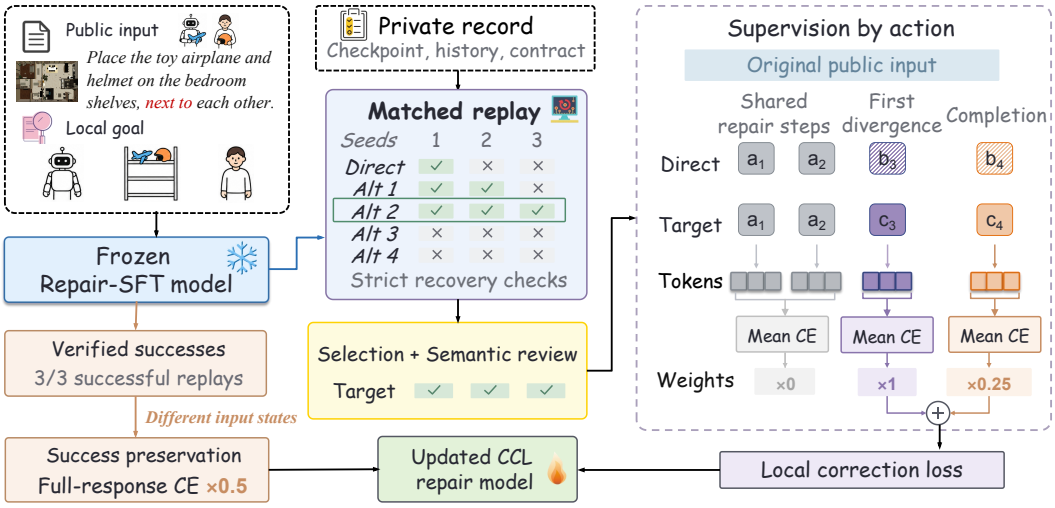


Figure 3: **Training from alternatives generated by the repair model.** shared steps are identical initial actions in the direct and accepted repairs. The diagram shows their masked loss and the primary losses on changed and later actions. Equation 3 also includes field and ending terms.

Because an episode may contribute several similar failures, states from the same episode and category form a group g and receive weight $1/|g|$. Normalizing these weights to mean one reduces the influence of episodes with many related events. The resulting parameters θ_{SFT} are then held fixed for candidate collection and initialize the CCL update (Figure 3). Thus witness supervision precedes learning from verified alternatives under the same interface.

3.4 Verifying Alternative Repair Programs

For each training input, the frozen model generates the direct repair without sampling and uses it as the source program y^-. It also samples four alternatives under the same prompt and call budget. Pairing requires a source that conforms to the public action schema. We replay the fixed outputs under a common set $\mathcal{Z}$ of three simulator seeds without further generation. Writing y^+ for an eligible alternative, the recovery count and acceptance conditions are

$$R(c, y) = \sum_{z \in \mathcal{Z}} V(c, y, z; \mathcal{C}), \qquad R(c, y^-) < 3, \quad R(c, y^+) = 3. \tag{1}$$

Here $R(c, y)$ counts successful replays from c: the alternative must pass all three while its source fails at least one.

To locate the changed decisions, we align JSON actions after normalizing key order and whitespace while retaining their agent, skill, and arguments. The *first divergence* has index d, and the $d-1$ identical initial calls form the shared repair steps, shown as shared steps in Figure 3. If one program stops while the other continues, that boundary also counts as a divergence. Let $h_z(c, y, j)$ hash the state fields recorded after j calls under seed z, with $j = 0$ denoting the restored state. Eligible pairs must satisfy

$$h_z(c, y^-, d-1) = h_z(c, y^+, d-1) \quad \text{for every } z \in \mathcal{Z}. \tag{2}$$

This comparison checks agreement on the recorded fields immediately before divergence, while Equation 1 checks recovery by the complete alternative.

When several alternatives satisfy both conditions, we select the one with the longest shared repair steps, followed by the fewest action insertions, deletions, or substitutions and then the shortest program. The sampling order and program identifier resolve any remaining ties. A human reviewer checks the selected program against the instruction and outcomes, accepting or rejecting it without rewriting its actions. The accepted triples $e = (x, y^-, y^+)$ form $\mathcal{D}_{\mathrm{CF}}$. Appendix E details this procedure.

3.5 Learning from Verified Repairs

Based on the alignment above, we place the primary loss on the first divergence because it marks the earliest changed decision under matching recorded conditions. Within this action, an additional term emphasizes the agent, skill, and argument values that specify its execution. The remaining accepted actions form the *completion*, which receives lower-weight supervision so that the model learns complete repairs while retaining the emphasis on the first divergence.

For a correction example e, let $u^+ = (u_1^+, \ldots, u_M^+)$ be the accepted program written as M tokens. The loss at token position t is $\ell_t(\theta; e) = -\log \pi_\theta(u_t^+ \mid x, u_{<t}^+)$, where $u_{<t}^+$ is the preceding target text. For a nonempty set $\mathcal{T}$ of supervised positions, write the mean token loss as $\bar{\ell}_{\mathcal{T}} = |\mathcal{T}|^{-1} \sum_{t \in \mathcal{T}} \ell_t$. To implement the supervision described above, the masks D, S, C, and E select the divergent action or stopping boundary, its semantic field values, the remaining completion, and the native response-ending token, respectively. Omitting empty components gives

$$\mathcal{L}_{\mathrm{CF}}(\theta; e) = \bar{\ell}_D + \bar{\ell}_S + \lambda_c \bar{\ell}_C + \lambda_e \bar{\ell}_E, \qquad \lambda_c = 0.25, \quad \lambda_e = 0.5. \tag{3}$$

Each component is averaged over its selected tokens before weighting to prevent length alone from increasing its contribution. Appendix F specifies the token alignment and reduction rules.

However, correction examples do not directly reinforce successful behavior on other states. We therefore construct $\mathcal{D}_{\mathrm{pres}}$ from verified source programs on training states disjoint from the correction set, using equally sized correction and preservation sets. Each preservation target must satisfy its fixed recovery contract, although its actions may differ from the construction witness. For a preservation example $s = (x_s, u_s)$, with public input x_s and N_s response tokens, we supervise the complete response:

$$\mathcal{L}_{\mathrm{pres}}(\theta; s) = -\frac{1}{N_s} \sum_{t=1}^{N_s} \log \pi_\theta(u_{s,t} \mid x_s, u_{s,<t}). \tag{4}$$

Here $u_{s,t}$ is the current token and $u_{s,<t}$ the preceding response text.

Before combining the two objectives, we weight examples to balance related failure events. Correction examples inherit the effective state weight b_e from Repair-SFT, which we multiply by the fraction of unsuccessful source replays, $\alpha_e = (3 - R(c, y^-))/3$. Normalizing these products to mean one over $\mathcal{D}_{\mathrm{CF}}$ yields ω_e. Preservation weights ν_s are independently normalized inverse group sizes within the selected preservation states. With these weights, we jointly optimize correction and preservation from θ_{SFT}:

$$\mathcal{L}_{\mathrm{CCL}}(\theta) = \frac{1}{|\mathcal{D}_{\mathrm{CF}}|} \sum_{e \in \mathcal{D}_{\mathrm{CF}}} \omega_e \mathcal{L}_{\mathrm{CF}}(\theta; e) + \frac{\lambda_p}{|\mathcal{D}_{\mathrm{pres}}|} \sum_{s \in \mathcal{D}_{\mathrm{pres}}} \nu_s \mathcal{L}_{\mathrm{pres}}(\theta; s), \quad \lambda_p = 0.5. \tag{5}$$

The coefficient λ_p controls the preservation contribution relative to correction. Appendix G gives the optimization settings. After training, the model generates a repair directly under the inference protocol in Appendix G.3.

As an illustration, Figure 4 revisits the training state in Figure 2, where the learned model places the airplane first and thus reverses the witness order while completing the same goal. Appendices J.3 and J.2 give the generated sequence and illustrative training pairs.

4 Experiments

We first compare local recovery on PARTNR-FS across models and repair methods, then test whether these gains benefit complete TDW-MAT tasks. To distinguish improvement from retention, we also separate new recoveries from losses on previously solved states.

4.1 Evaluation Setup

Tables 1 and 2 evaluate all 183 test states under the same public input fields and five-call limit. Strict recovery requires a fixed program to pass all three replay seeds. Program validity measures compliance with parsing and action-schema checks, while model calls per request quantify inference cost. Header n gives subset sizes. Appendix H defines the metrics.



(a) Airplane held. (b) Helmet held. (c) Recovered.

Figure 4: **Repairing a placement involving both agents.** Qwen3-8B CCL places the airplane first and the helmet beside it in a four-call repair of training episode 1425. The seed-17 replay closes the local placement goal.

We apply this protocol to Qwen2.5-7B (Yang et al., 2025b), Llama3.1-8B (Grattafiori et al., 2024; Meta AI, 2024), Qwen3-8B (Yang et al., 2025a; Team, 2025), and Qwen3.5-9B (Qwen Team, 2026), comparing each published model with Repair-SFT and CCL. Both training stages use low-rank adaptation (LoRA) (Hu et al., 2021), and each backbone collects alternatives with its own frozen Repair-SFT model. Prompted GPT-4o (Hurst et al., 2024), Gemini 3.1 Pro (Google DeepMind, 2026), Grok 4.5 (SpaceXAI, 2026), and GPT-5.6 Sol (OpenAI, 2026) provide additional references under the same local evaluation protocol. Appendix G separates the model-training settings from the replay seeds used to execute fixed outputs.

4.2 Recovery across Different Backbones

Table 1 shows that CCL improves strict recovery over Repair-SFT across all four backbones, with gains of 2.74–9.29 percentage points. These gains follow the substantial improvement already obtained by Repair-SFT, indicating a further benefit from training on verified alternatives generated by each model. At the same time, it can be observed that as the backbone's capabilities improve, the performance gains resulting from CCL become increasingly larger.

The category breakdown in Table 1 shows reachability and holding gains for every model. Qwen2.5-7B matches Repair-SFT on unsupported actions and relations, whereas the other models improve. Figure 5 highlights the gains across all four categories for Qwen3.5-9B.

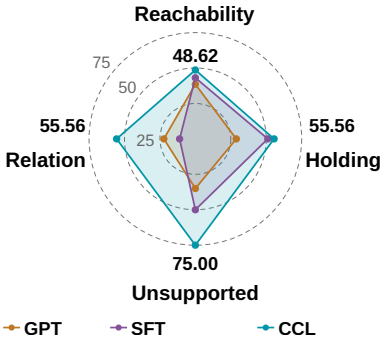


Figure 5: **Category recovery (%).**

4.3 Comparison with Repair Methods

Table 2 compares CCL with other repair methods under the same failure inputs. ProgPrompt expresses available skills as programs, while Self-Refine and CMFR refine a proposed repair through feedback. CSO and Agent-R learn from verified decisions or revised trajectories. Public Heuristic supplies a rule-based reference. All are adapted to the local program interface described in Appendix H.2, with refinement calls included in the reported inference cost.

CCL reaches 47.54% strict recovery, exceeding Agent-R, the strongest baseline overall, by 4.37 percentage points. This difference is largest on unsupported actions, with additional gains on holding and matching scores on relations. Public Heuristic produces valid programs for every state but recovers fewer than CCL, showing that legality alone does not ensure task recovery. CCL achieves this recovery rate with one model call per request and no simulator search at inference.

4.4 Complete Cooperative Tasks on TDW-MAT

To test whether local repairs support continued cooperative progress, we evaluate complete transport tasks in ThreeDWorld (Gan et al., 2020) with a 3,000-frame budget. Table 3 compares RHP and

Table 1: **Recovery across different backbones on PARTNR-FS.** Scores are strict recovery rates (%) on 183 test states, requiring success under all replay seeds. Parentheses give absolute percentage-point gains over the corresponding Repair-SFT model, calculated from the displayed values.

Method	Overall $n = 183$	Reach $n = 109$	Hold $n = 45$	Unsupported $n = 20$	Relation $n = 9$
GPT-4o	9.84	9.17	11.11	5.00	22.22
Gemini 3.1 Pro	21.31	23.85	17.78	15.00	22.22
Grok 4.5	26.78	26.61	24.44	30.00	33.33
GPT-5.6 Sol	34.97	38.53	28.89	35.00	22.22
Qwen2.5-7B	1.09	0.00	4.44	0.00	0.00
+ Repair-SFT	37.70	35.78	46.67	40.00	11.11
+ CCL	40.44 (+2.74)	38.53 (+2.75)	51.11 (+4.44)	40.00 (+0.00)	11.11 (+0.00)
Llama3.1-8B	0.00	0.00	0.00	0.00	0.00
+ Repair-SFT	36.61	40.37	35.56	35.00	0.00
+ CCL	43.72 (+7.11)	44.04 (+3.67)	46.67 (+11.11)	50.00 (+15.00)	11.11 (+11.11)
Qwen3-8B	4.92	3.67	8.89	5.00	0.00
+ Repair-SFT	39.89	40.37	37.78	50.00	22.22
+ CCL	47.54 (+7.65)	42.20 (+1.83)	48.89 (+11.11)	75.00 (+25.00)	44.44 (+22.22)
Qwen3.5-9B	5.46	2.75	6.67	5.00	33.33
+ Repair-SFT	44.26	43.12	51.11	50.00	11.11
+ CCL	53.55 (+9.29)	48.62 (+5.50)	55.56 (+4.45)	75.00 (+25.00)	55.56 (+44.45)

Table 2: **Method comparison on PARTNR-FS.** Scores are strict recovery rates (%) on 183 test states with unconstrained outputs. Calls denote model invocations per context. CCL and its Repair-SFT initialization use Qwen3-8B.

Method	Overall $n = 183$	Reach $n = 109$	Hold $n = 45$	Unsup. $n = 20$	Relation $n = 9$	Valid $n = 183$	Calls
Public Heuristic	24.59	24.77	28.89	25.00	0.00	100.00	0
ProgPrompt (Singh et al., 2022)	9.29	7.34	17.78	5.00	0.00	61.75	1
Self-Refine (Madaan et al., 2023)	4.92	1.83	13.33	5.00	0.00	47.54	3
CMFR (Farag et al., 2025)	10.93	11.93	11.11	5.00	11.11	73.22	3
CSO (Li et al., 2026a)	42.08	40.37	42.22	60.00	22.22	97.81	1
Agent-R (Yuan et al., 2025)	43.17	41.28	42.22	55.00	44.44	90.16	1
CCL	**47.54**	42.20	48.89	75.00	44.44	92.35	1

CoELA (Zhang et al., 2024), REVECA (Seo et al., 2025), CaPo (Liu et al., 2025), and PCE (Seo et al., 2026), with SFT and CCL repair modules added to CoELA and PCE. The repair modules use Qwen3.5-9B, with the CCL update trained on the TDW-MAT training split. With CoELA, GPT-4o retains planning, the partner continues to act during repair, and control returns to CoELA afterward. Final transport rate measures the endpoint, while TR-AUC integrates progress over the common frame budget. Appendix H.3 describes the baseline controllers and settings.

Adding CCL to CoELA improves its overall transport rate by 10.00 percentage points and TR-AUC by 8.64 points. Relative to CoELA (w/ SFT), the corresponding gains are 5.42 and 4.26 points, indicating an additional benefit beyond supervised repair initialization. Figure 6 shows the CoELA progress curves under the common frame budget.

The benefit also holds for the stronger PCE controller: PCE (w/ CCL) exceeds PCE (w/ SFT) by 2.50 points in transport rate and 2.18 points in TR-AUC. It achieves the highest values on both metrics in Table 3, with final transport gains on both Food and Stuff.

4.5 Recovery Gains and Retained Behavior

Figure 7 separates recovery gains from regressions on previously solved states. Gains exceed losses for every model, although none retains every previously solved state. Qwen3-8B retains 67 of 73

Table 3: **Complete task evaluation on TDW-MAT.** Food, Stuff, and Overall report final transport rates. TR-AUC measures transport progress over the common frame budget.

System	Transport rate ↑			TR-AUC ↑
	Food	Stuff	Overall	Overall
RHP	72.50	75.83	74.17	27.37
REVECA	83.33	75.00	79.17	40.05
CaPo	80.83	78.06	79.44	33.96
CoELA	76.67	71.67	74.17	29.65
PCE	88.33	85.83	87.08	42.82
CoELA (w/ SFT)	80.00	77.50	78.75	34.03
CoELA (w/ CCL)	85.00	83.33	84.17	38.29
PCE (w/ SFT)	89.17	85.83	87.50	43.25
PCE (w/ CCL)	**91.67**	88.33	90.00	45.43

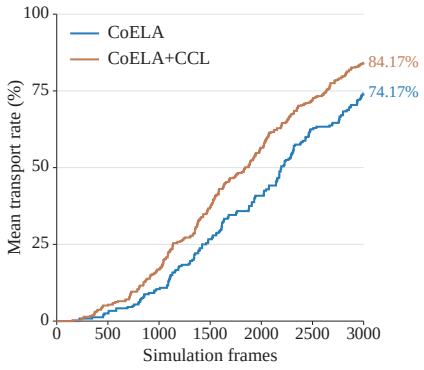
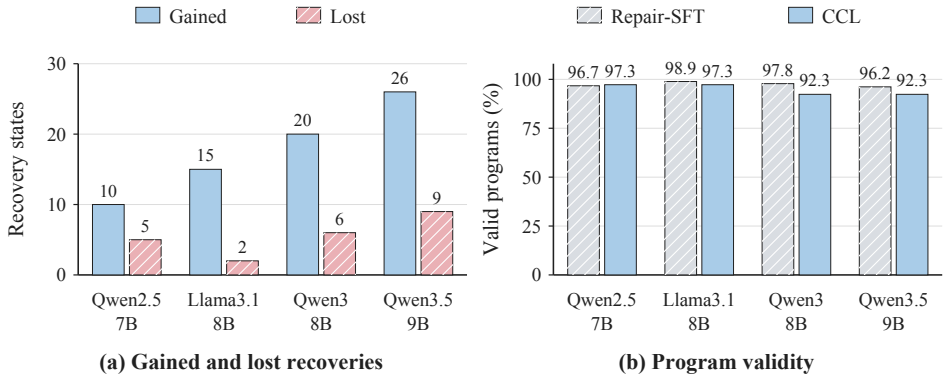


Figure 6: **TDW-MAT progress.** Mean transport rate over common frame budget.

Repair-SFT successes. Validity reveals a related trade-off: Qwen2.5 improves both metrics, whereas Qwen3 and Qwen3.5 recover more states while still producing invalid outputs. Appendix I.1 adds a Full-response CE control (43.90% recovery versus 47.54% for CCL). Appendix I also reports paired outcomes and collection cost.



(a) Gained and lost recoveries **(b) Program validity**

Figure 7: **Recovery gains and retained behavior.** Gained and lost states relative to Repair-SFT, alongside program validity. The difference between gained and lost states equals net recovery gain.

5 Conclusion

In this work, we introduced Counterfactual Contract Learning (CCL) for local execution recovery in embodied multi-agent collaboration. It identifies repair programs that complete the local goal while preserving previously satisfied conditions. Program alignment then guides supervision toward the first differing action, while lower-weight supervision on the continuation and termination teaches complete repairs. This approach improves strict recovery across four backbones and outperforms the compared local repair methods. Experiments on TDW-MAT further show that these improvements can support cooperative task progress and final object transport.

We identify three limitations of our work that suggest directions for future research. First, CCL relies on successful alternatives obtained through sampling, motivating more effective candidate generation to cover a broader range of failure states. Second, the actor-indexed program representation provides a basis for extending our evaluation to teams with different sizes and agent capabilities. Finally, occasional regressions on previously solved states suggest exploring a more adaptive balance between correction and preservation.

AI USE STATEMENT

AI tools assisted with language polishing, dataset generation, and experimental testing, including the generation of candidate repair programs. Semantic review was performed by human reviewers. The authors retain responsibility for the research methods, experimental records, interpretations, references, and final manuscript.

ETHICS STATEMENT

This work studies reliable execution recovery in embodied collaboration. Recovery targets are checked against task goals and previously satisfied conditions, while private verification records remain separate from model inputs. Use and redistribution of third-party software, models, and environment assets are subject to their respective access conditions and licenses. Practical deployment should include application-specific safety checks and human oversight.

REPRODUCIBILITY STATEMENT

The supplementary materials provide code for PARTNR-FS dataset construction, Repair-SFT, CCL training, and local repair evaluation, together with backbone-specific LoRA configurations, protocol tests, and recorded repair demonstrations. The appendix describes the dataset partitions, public action interface, candidate generation, training settings, and evaluation protocols. It also details the regional objective, supervision controls, baseline adaptations, and qualitative examples distinguishing construction witnesses from repairs generated after training.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

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

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Akash Bonagiri, Devang Borkar, Gerard Janno Anderias, Setareh Rafatirad, and Houman Homayoun. Causalflow: Causal attribution and counterfactual repair for llm agent failures. *arXiv preprint arXiv:2605.25338*, 2026.

Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, et al. Partnr: A benchmark for planning and reasoning in embodied multi-agent tasks. In *International Conference on Learning Representations*, volume 2025, pp. 65205–65268, 2025.

Zi-Yi Dou, Cheng-Fu Yang, Xueqing Wu, Kai-Wei Chang, and Nanyun Peng. Re-rest: Reflection-reinforced self-training for language agents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 15394–15411, 2024.

Youmna Farag, Svetlana Stoyanchev, Mohan Li, Simon Keizer, and Rama Doddipatla. Conditional multi-stage failure recovery for embodied agents. In *Proceedings of the 1st Workshop for Research on Agent Language Models (REALM 2025)*, pp. 200–227, 2025.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma Gongque, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. Agentrefine: Enhancing agent generalization through refinement tuning. In *International Conference on Learning Representations*, volume 2025, pp. 65185–65204, 2025.

Chuang Gan, Jeremy Schwartz, Seth Alter, Damian Mrowca, Martin Schrimpf, James Traer, Julian De Freitas, Jonas Kubilius, Abhishek Bhandwaldar, Nick Haber, et al. Threedworld: A platform for interactive multi-modal physical simulation. *arXiv preprint arXiv:2007.04954*, 2020.

Heyang Gao, Zexu Sun, Erxue Min, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Xu Chen. Solving the granularity mismatch: Hierarchical preference learning for long-horizon llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 67081–67110, 2026.

Google DeepMind. Gemini 3.1 Pro: Model card, 2026. URL `https://deepmind.google/models/model-cards/gemini-3-1-pro/`.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 54523–54549, 2025.

Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li E Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37: 100428–100534, 2024.

Mukai Li, Qingcheng Zeng, Tianqing Fang, Zhenwen Liang, Linfeng Song, Qi Liu, Haitao Mi, and Dong Yu. Verified critical step optimization for llm agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 39627–39639, 2026a.

Zichao Li, Gang Wu, Jack Wang, Ruiyi Zhang, Wanrong Zhu, Ryan Rossi, Vlad Morariu, and Jihyung Kil. Spinning straw into gold: Relabeling llm agent trajectories in hindsight for successful demonstrations. In *International Conference on Learning Representations*, volume 2026, pp. 148186–148209, 2026b.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Jie Liu, Pan Zhou, Yingjun Du, Ah-Hwee Tan, Cees G Snoek, Jan-Jakob Sonke, and Efstratios Gavves. Capo: Cooperative plan optimization for efficient embodied multi-agent cooperation. In *International Conference on Learning Representations*, volume 2025, pp. 70791–70815, 2025.

Zeyi Liu, Arpit Bahety, and Shuran Song. Reflect: Summarizing robot experiences for failure explanation and correction. *arXiv preprint arXiv:2306.15724*, 2023.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Dover: Intervention-driven auto debugging for llm multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 28059–28087, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Meta AI. Llama-3.1-8B-Instruct: Model card, 2024. URL https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *International conference on machine learning*, pp. 3878–3887. PMLR, 2018.

OpenAI. GPT-5.6 Sol: Model documentation, 2026. URL https://developers.openai.com/api/docs/models/gpt-5.6-sol.

Aishwarya Padmakumar, Jesse Thomason, Ayush Shrivastava, Patrick Lange, Anjali Narayan-Chen, Spandana Gella, Robinson Piramuthu, Gokhan Tur, and Dilek Hakkani-Tur. Teach: Task-driven embodied agents that chat. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 2017–2025, 2022.

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. *arXiv preprint arXiv:2010.09890*, 2020.

Xavier Puig, Eric Undersander, Andrew Szot, Mikael Dallaire Cote, Tsung-Yen Yang, Ruslan Partsey, Ruta Desai, Alexander Clegg, Michal Hlavac, So Yeon Min, et al. Habitat 3.0: A co-habitat for humans, avatars, and robots. In *International Conference on Learning Representations*, volume 2024, pp. 15306–15336, 2024.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. *Advances in neural information processing systems*, 32, 2019.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9338–9346. IEEE, 2019.

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

SeungWon Seo, SooBin Lim, SeongRae Noh, Haneul Kim, and HyeongYeop Kang. From assumptions to actions: Turning llm reasoning into uncertainty-aware planning for embodied agents. *arXiv preprint arXiv:2602.04326*, 2026.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 60808–60838, 2025.

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. Rl's razor: Why online reinforcement learning forgets less. In *International Conference on Learning Representations*, volume 2026, pp. 59839–59864, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *2020 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 10737–10746. IEEE, 2020a.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020b.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 2998–3009, 2023.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7584–7600, 2024.

SpaceXAI. Introducing Grok 4.5, 2026. URL `https://x.ai/news/grok-4-5`.

Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, Karen Liu, et al. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments. In *Conference on robot learning*, pp. 477–490. PMLR, 2022.

Qwen Team. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

Hanlin Wang, Chak Tou Leong, Jian Wang, and Wenjie Li. E2cl: exploration-based error correction learning for embodied agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7626–7639, 2024.

Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. Steca: Step-level trajectory calibration for llm agent learning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11597–11614, 2025.

Weimin Xiong, Yifan Song, Xiutian Zhao, Wenhao Wu, Xun Wang, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. Watch every step! llm agent learning via iterative step-level process refinement. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1556–1572, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Matthew Yang, Hao Bai, Ian Wu, Gene Yang, Amrith Setlur, and Aviral Kumar. Int: Self-proposed interventions enable credit assignment in llm reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 85054–85091, 2026.

Qwen: An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025b. URL `https://arxiv.org/abs/2412.15115`.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training. *arXiv preprint arXiv:2501.11425*, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

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*, volume 2024, pp. 19373–19401, 2024.

Hongxin Zhang, Zeyuan Wang, Qiushi Lyu, Zheyuan Zhang, Sunli Chen, Tianmin Shu, Behzad Dariush, Kwonjoon Lee, Yilun Du, and Chuang Gan. Combo: Compositional world models for embodied multi-agent cooperation. In *International Conference on Learning Representations*, volume 2025, pp. 49996–50019, 2025.

Qinhong Zhou, Chuang Gan, and Anoop Cherian. Llawco: Learning laws of cooperation for modeling embodied multi-agent behavior. *arXiv preprint arXiv:2606.28182*, 2026.

Appendix

Table of Contents

A Mathematical Notation **16**

B Dataset Construction and Provenance **18**
- B.1 Local execution events and recovery witnesses 18
- B.2 Dataset composition and partitions 18
- B.3 Failure families and correlated events 19

C Public Inputs and the Action Interface **19**
- C.1 Policy inputs and verification records 19
- C.2 Actor capabilities and program syntax 20
- C.3 Message construction 20

D Checkpoint Replay and Recovery Verification **20**
- D.1 Restoring and matching execution states 20
- D.2 Contract evaluation 21
- D.3 Execution horizon and semantic review 21

E Counterfactual Collection and Preservation **21**
- E.1 Generating programs with the frozen repair model 21
- E.2 Action alignment and candidate selection 22
- E.3 Independent success preservation 23

F Regional Objective and Token Alignment **23**
- F.1 Supervision components 23
- F.2 Token mapping and stopping boundaries 24
- F.3 Sample normalization and gradient accumulation 24

G Training and Repair Inference **24**
- G.1 Backbones and supervised initialization 24
- G.2 Optimization settings 25
- G.3 Generation, parsing, and execution 25

H Evaluation Protocols and Baseline Adaptations **25**
- H.1 Local recovery and program validity 25
- H.2 Local adaptations of repair methods 26
- H.3 TDW-MAT baselines 26

I Supervision Controls and Additional Analysis **27**
- I.1 Supervision scope 27
- I.2 Completion coefficient 28
- I.3 Paired recovery outcomes 28
- I.4 Collection and deployment workload 28

J Qualitative Records and Executed Repairs **29**
- J.1 Representative PARTNR-FS records 29
- J.2 Verified repair-program pairs 31
- J.3 A repair generated by the trained policy 32

A MATHEMATICAL NOTATION

Table A.1 collects the symbols used in the method and appendix. The recovery contract $\mathcal{C}$ is distinct from completion mask C, and evaluation set $\mathcal{S}$ from semantic-field mask S. Program roles, action positions, and token positions are distinguished below.

Table A.1: **Key mathematical notation.**

Notation	Definition
Programs and execution records	
$x,\ x_s$	Public input to the repair model; the subscript s identifies a preservation example.
$c,\ c_i$	Private replay record containing the simulator checkpoint and task history; c_i belongs to test state i.
$\mathcal{C},\ \mathcal{C}_i$	Fixed recovery contract defining the local goal and protected conditions; $\mathcal{C}_i$ belongs to test state i.
$\theta,\ \theta_{\mathrm{SFT}}$	Trainable model parameters and the parameters obtained after Repair-SFT.
$\pi_\theta(y \mid x)$	Conditional probability assigned by the repair model to program y given public input x.
$y = (a_1, \ldots, a_T)$	Repair program comprising an ordered sequence of skill calls.
$T,\ j$	Program length and skill-call position. The complete program satisfies $T \leq 5$.
$a_j = (i_j, k_j, p_j)$	Skill call at position j, comprising an acting agent, a skill, and an argument list.
$i_j,\ k_j,\ p_j$	Identifier of the executing agent in the public interface, skill name, and ordered argument list.
$y^-,\ y^+$	Direct repair used as the source and a verified alternative used as the correction target. Superscripts identify their roles.
Replay, alignment, and training collections	
$\mathcal{Z},\ z$	Replay seed set $\{17, 29, 43\}$ and one seed in that set.
$V(c, y, z; \mathcal{C})$	Binary verifier result: one when replay of y from c under seed z passes the checks specified by $\mathcal{C}$, and zero otherwise.
$R(c, y)$	Number of successful replays of y, summed over $\mathcal{Z}$.
$d,\ d-1$	First differing action or stop/continue boundary, and the number of identical initial calls preceding it.
$h_z(c, y, j)$	Hash of the recorded state fields after j calls under seed z; $j = 0$ denotes the restored initial state.
$e = (x, y^-, y^+)$	Accepted correction example containing public input and paired source and target programs.
$\mathcal{D}_{\mathrm{CF}},\ N_{\mathrm{CF}}$	Set of accepted correction examples and its size, $N_{\mathrm{CF}} = \lvert\mathcal{D}_{\mathrm{CF}}\rvert$.
$s = (x_s, u_s)$	Preservation example containing public input and a complete verified response.
$\mathcal{D}_{\mathrm{pres}},\ N_{\mathrm{pres}}$	Set of preservation examples and its size, $N_{\mathrm{pres}} = \lvert\mathcal{D}_{\mathrm{pres}}\rvert$. Its states are disjoint from those in $\mathcal{D}_{\mathrm{CF}}$.
$\mathcal{P}$	Pool of verified successful direct repairs used to select preservation examples.
Token supervision and losses	
$u^+,\ M,\ t$	Accepted response as a token sequence, its length M, and target-token position t.
$u_t^+,\ u_{<t}^+$	Target token at position t and all target tokens preceding it.
$u_s,\ N_s$	Verified preservation response and its token count; N_s is a response length, not a collection size.
$u_{s,t},\ u_{s,<t}$	Current target token and all preceding tokens in preservation response u_s.
i (token alignment)	Position in the rendered training sequence; the prediction at $i-1$ is aligned with target token i.
$\ell_t(\theta; e)$	Token negative log likelihood, $-\log \pi_\theta(u_t^+ \mid x, u_{<t}^+)$.
$\mathcal{T},\ \lvert\mathcal{T}\rvert$	Mask represented as a set of supervised token positions, and the number of positions it contains.
$\bar{\ell}_{\mathcal{T}}(\theta; e)$	Mean token loss over mask $\mathcal{T}$; an intentionally empty mask contributes zero.

Continued on the next page.

Table A.1 (continued)

Notation	Definition
D	Mask for the complete first divergent action or target stopping boundary.
S	Mask for agent, skill, and argument values in the divergent action; it overlaps D.
C	Completion mask for accepted actions after the divergence. This mask is distinct from recovery contract $\mathcal{C}$.
E	Mask for the native response-ending token; contributions may overlap other masks.
$\lambda_c,\ \lambda_e,\ \lambda_p$	Loss coefficients for completion, ending, and preservation; the CCL setting uses 0.25, 0.5, and 0.5, respectively.
$\mathcal{L}_{\mathrm{CF}}(\theta; e)$	Correction loss combining independently averaged divergence, semantic-field, completion, and ending losses.
$\mathcal{L}_{\mathrm{pres}}(\theta; s)$	Mean negative log likelihood over the complete preservation response.
$\mathcal{L}_{\mathrm{CCL}}(\theta)$	Training objective combining weighted correction and preservation means.
Example weighting and evaluation	
$g,\ \lvert g\rvert$	Group of training states from the same episode and failure category, and the group size used for inverse weighting.
b_e	Effective state weight inherited by correction example e from the Repair-SFT training data.
α_e	Fraction of unsuccessful source replays, $(3 - R(c, y^-))/3$.
$g_{\mathrm{pres}}(s),\ \tilde{b}_s$	Group of selected preservation examples sharing an episode and category with s, and raw weight $\tilde{b}_s = 1/\lvert g_{\mathrm{pres}}(s)\rvert$.
$\omega_e,\ \nu_s$	Correction and preservation weights normalized separately to mean one over their respective collections.
$e',\ s'$	Summation indices over correction and preservation examples in the weight-normalization denominators.
$\mathcal{S},\ i$ (evaluation)	Test-state index set and an index in that set. The full local evaluation uses $\lvert\mathcal{S}\rvert = 183$; this set differs from mask S.
$y_i,\ r_i$	Generated program for test state i and its strict-success indicator, equal to one only when all designated replays pass.
$r_i^{\mathrm{SFT}},\ r_i^{\mathrm{CCL}}$	Strict-success indicators for the fixed Repair-SFT and CCL outputs on test state i.
$n_{\mathrm{keep}},\ n_{\mathrm{gain}}$	Counts of test states solved by both models and solved only after CCL, respectively.
$n_{\mathrm{lost}},\ n_{\mathrm{fail}}$	Counts of test states solved only before CCL and solved by neither model, respectively.
n (table headers)	Number of evaluated states in the indicated full set or failure-category subset.
$\mathrm{StrictRecovery}$	Percentage of test states for which the fixed output succeeds under every designated replay seed.
$\Delta\mathrm{StrictRecovery}$	CCL strict recovery rate minus the Repair-SFT rate, measured in percentage points.
$\mathrm{Retention}$	Fraction $n_{\mathrm{keep}}/(n_{\mathrm{keep}} + n_{\mathrm{lost}})$ of initial successes retained, when the denominator is positive.
p (top-p)	Cumulative-probability threshold for alternative sampling, fixed at 0.9. It differs from action argument list p_j.
β	Preference-logit scale in the CSO-Local DPO baseline, fixed at 0.5; not a CCL loss coefficient.
$\lvert\cdot\rvert,\ \varnothing,\ \leftarrow$	Set cardinality, the empty set, and assignment in the collection algorithm.

B Dataset Construction and Provenance

ALFRED and ALFWorld study instruction following (Shridhar et al., 2020a;b), while BEHAVIOR defines household activities through initial and goal conditions (Srivastava et al., 2022). TEACh and Watch-And-Help extend evaluation to dialogue and collaborative assistance (Padmakumar et al., 2022; Puig et al., 2020), and Embodied Agent Interface tests decision-making components (Li et al., 2024). PARTNR-FS focuses on recovery within such tasks by retaining the execution state and local goal at each failure event.

B.1 Local execution events and recovery witnesses

We collect local failure events from two-agent PARTNR executions using GPT-4o with ReAct (Chang et al., 2025; Yao et al., 2022; Hurst et al., 2024). Each event retains the public instruction, observations, feedback, and recent history, together with a separate simulator checkpoint and task history for replay. An episode may contribute several events at different stages; the direct repairs used for CCL are generated later by the frozen Repair-SFT model.

Recovery contract construction. For each recorded failure, we construct a recovery contract from the official task predicates and temporal dependencies, interpreted against the saved execution state and task history. Automated processing links objects involved in the failed action, currently held objects, and task objects in relevant containers to candidate predicates. Human reviewers use this evidence to select the current local responsibility and bind its conditions to the relevant objects and task stage. This distinguishes physical conditions that must hold after repair from earlier milestones that need only have been completed, with object-release and prerequisite requirements added where applicable. Discrepancies between the instruction and available predicates are flagged for review. When several outcomes are valid, the contract admits alternative branches: all conditions in at least one branch must be satisfied. The reviewed contract is frozen before candidate testing and specifies acceptable outcomes rather than a repair sequence. Replay checks these conditions alongside separate checks for task-state regression; a verified witness then establishes that the fixed contract can be satisfied within the five-call budget. The witness must pass replay and semantic review, providing both evidence of local recoverability and a target for supervised initialization. Section J.1 gives examples from all four failure categories, including the placement prerequisite in Figure 2.

B.2 Dataset composition and partitions

Table B.1 reports the dataset partitions and their scene and episode coverage. Training states supply the initialization witnesses and contexts for collecting alternatives; both local main-paper comparisons use the full test partition.

Table B.1: **PARTNR-FS composition.** Episode counts refer to distinct scene–episode pairs. The total scene count is deduplicated across partitions.

Split	Reach	Hold	Unsupported	Relation	States	Scenes	Episodes
Train	256	115	53	33	457	32	318
Validation	56	31	23	6	116	6	83
Test	109	45	20	9	183	36	104
Total	421	191	96	48	756	74	505

The partitions are disjoint in context identifiers, checkpoint hashes, scene–episode pairs, and exact scene–instruction pairs.

Figure B.1 shows training witnesses spanning one to five calls, with two-call programs most frequent. Call count measures program length rather than duration because a high-level skill may require several simulator steps.

B.3 Failure families and correlated events

Reachability covers distance or access conditions needed to complete a call. *Holding* covers conflicts between the held-object state and an action preconditions. *Unsupported action* covers a call unavailable to the actor, interface, or target entity, including an opening request on non-articulated furniture. *Relation* concerns a required spatial relation or its arguments.

A record retains its assigned primary category when the feedback mentions several contributing conditions; Figure B.2 shows the overall distribution.

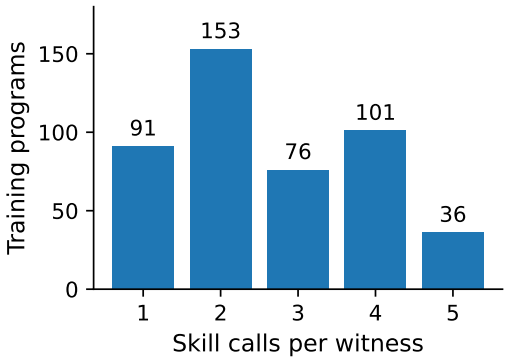


Figure B.1: **Training witness lengths.** Counts of the 457 verified programs by number of high-level skill calls.

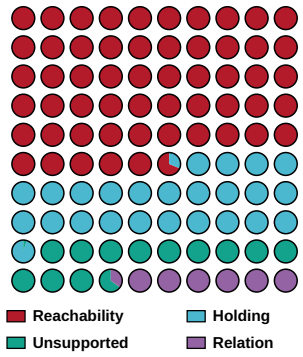


Figure B.2: **Failure-family distribution.** Each circle denotes 1% of the 756 recovery states; split circles retain fractional shares.

These categories describe execution mechanisms. Actor and object fields retain the context needed to interpret a partner dependency: a holding failure can involve an object held by the acting agent or by its partner. Section 3.3 groups correlated events by episode and mechanism. Counterfactual learning inherits their normalized effective weights, whereas the preservation stream recomputes group sizes on its selected contexts (Section F.3).

C Public Inputs and the Action Interface

C.1 Policy inputs and verification records

The shared input supplies the instruction, failure feedback, available skills, recent history, and public agent and object states. Public skill signatures appear in `action_contract`; the private `responsibility_contract` records the recovery goal and protected conditions in $\mathcal{C}$. Table C.1 separates these conditioning inputs from training targets and verification records.

Table C.1: **Information roles.** Supervision is supplied as an assistant response, while verification uses separate records.

Information	Policy prompt	Use during learning
Instruction, public state, feedback, history	Included	Conditioning context
Actor capabilities and action signatures	Included	Output interface
Full checkpoint and evaluation history	Excluded	Replay and verification
Responsibility contract and task predicates	Excluded	Target acceptance
Verified construction witness	Excluded	Repair-SFT target
Accepted alternative program	Excluded	Regional correction target
Disjoint verified direct repair	Excluded	Preservation target
Replay outcomes and semantic reviews	Excluded	Data selection

Source generation, alternative sampling, and training reuse the same public input. Teacher forcing adds the accepted response shared repair steps as context for subsequent tokens, while simulator endpoints and review decisions remain outside the prompt.

C.2 Actor capabilities and program syntax

In the record interface, the actor identifies the embodied agent assigned to a call. The output is a JSON array containing between one and five action objects. Each object has exactly three fields: `agent` is a string actor identifier, `skill` names a callable skill, and `args` is a list of string arguments. The experiments instantiate this interface with two agents; Table C.2 gives their capability split. The central repair program assigns successive calls to available actors, and the skill driver executes them in their listed order. The budget counts calls across the entire program, including `Wait`.

Table C.2: **PARTNR-FS actor capabilities.** Actor 1 supports the shared skill set and additional state-changing skills.

Applicable actors	Callable skills
Actors 0 and 1	`Close`, `Explore`, `Navigate`, `Open`, `Pick`, `Place`, `Rearrange`, `Wait`
Actor 1 additionally	`Clean`, `Fill`, `Pour`, `PowerOff`, `PowerOn`

Single-entity calls take one referenced entity; `Explore` takes a room, and `Pick`, `Fill`, and `Pour` take an object. `Wait` uses an empty argument list. Placement and rearrangement take `[object, relation, receptacle]`, optionally extended with `next_to` and a reference object. The placement relation is one of `on`, `in`, `inside`, or `within`; optional fields are omitted rather than filled with a `None` string. The compiler checks actor capabilities, entity references, argument arity, and the total call budget. `Rearrange` is a high-level environment skill and may expand into several physical operations.

C.3 Message construction

Repair-SFT, candidate generation, and direct evaluation share the public program interface. The supplied system instruction is:

A complete Repair-SFT target. For training episode 1425, the task is to place the toy airplane and helmet next to each other on the bedroom shelves. Agent 0 (Alice) holds the airplane, and agent 1 (Bob) holds the helmet. The assistant response used as the verified construction witness is:

```
[
  {"agent":"1","skill":"Navigate","args":["shelves_7"]},
  {"agent":"1","skill":"Place",
   "args":["helmet_1","on","shelves_7"]},
  {"agent":"0","skill":"Navigate","args":["shelves_7"]},
  {"agent":"0","skill":"Place",
   "args":["toy_airplane_0","on","shelves_7",
           "next_to","helmet_1"]}
]
```

The user message gives the action schema followed by the public state under `STATE=`. Repair-SFT uses the verified witness as the assistant response, whereas candidate collection generates responses from the frozen model without providing a witness or earlier candidate. Both use the native chat template of each backbone and require a JSON program without explanatory text.

Bob first places the helmet, enabling Alice to place the airplane beside it. This JSON is the supervised response rather than part of the public input; the native chat template supplies its ending token. Complete prompts accompany the supplementary material, and case C4 illustrates a repair generated after training that uses the reverse object order.

D Checkpoint Replay and Recovery Verification

D.1 Restoring and matching execution states

Replay restores the checkpoint associated with each fixed program, including agent poses, held objects, rigid and articulated objects, and the public graph. The evaluation history is either reconstructed or

supplied by a certified checkpoint-state record, preserving the meaning of earlier completed stages. Source and alternative share this checkpoint and evaluation context within each of seeds 17, 29, and 43.

The trace records a hash of the initial state and after every high-level call. For divergence index d, we compare the hashes after the shared $d-1$ calls, or the initial hashes when $d=1$, as required by Equation 2. This comparison uses canonical serialization of the recorded fields; checkpoint restoration separately uses its numerical tolerance.

D.2 Contract evaluation

Verification checks program validity and checkpoint restoration before comparing the initial and final task records against the fixed contract. Object support, held-object status, and spatial or temporal predicates establish whether the local goal is complete and protected conditions remain satisfied. Strict recovery requires all designated recovery flags to pass and the recurrence flag to be false under every replay seed. A legal sequence can therefore fail recovery even when its actions execute, as case C3 in Section J.2 illustrates.

D.3 Execution horizon and semantic review

The central skill driver executes the fixed program without language-model calls or additional post-repair continuation. Recovery and recurrence are evaluated over this local execution window, while TDW-MAT separately measures complete cooperative tasks.

After automatic replay verification, human reviewers compare each selected training program and its recorded physical endpoint with the instruction and protected task conditions. They accept or reject the original program without rewriting any action. Semantic review is conducted manually, without a language model making the acceptance decision. Parsing failures remain model errors; infrastructure exceptions are recorded separately and rerun with the same fixed program rather than a newly generated repair.

E Counterfactual Collection and Preservation

Counterfactual collection uses only training states and a frozen Repair-SFT checkpoint. For each state, the same model generates one greedy source program and four sampled alternatives from identical public inputs. We fix these programs before replaying each from the stored checkpoint using the same recovery contract under 3 seeds. A correction pair requires a schema-valid source that succeeds in fewer than three replays and an alternative that succeeds in all three. We locate their first differing action immediately before that boundary under every replay seed. Among eligible alternatives, we prefer the longest common action prefix, then the smallest action edit distance and shortest target program, with deterministic tie-breaking. The selected pair undergoes case-by-case semantic review before admission. The collections are frozen before the CCL update; private contracts, replay outcomes, and review records are excluded from policy inputs. This procedure identifies verified alternative repairs and their earliest decision differences, rather than establishing that the first difference alone causes the recovery.

E.1 Generating programs with the frozen repair model

Each frozen Repair-SFT model generates one direct repair and four sampled alternatives from the same public context and call budget. Table E.1 gives the generation and replay settings; sampling is deterministically seeded for each context.

A source that passes parsing and public-schema checks can supply a correction pair when an alternative succeeds in all three replays and the source does not. Sources already successful in all three enter the preservation pool instead. We fix the generated outputs before attaching replay outcomes.

Source of the target. Collecting behavior from the learner connects to the treatment of learner-induced states in DAgger and the use of student-generated sequences in GKD (Ross et al., 2011;

Table E.1: **Program generation and replay.** Generation seeds control model sampling; replay seeds control execution of a fixed program.

Setting	Direct repair / evaluation	Sampled alternative
Decoding	Greedy	Stochastic
Temperature	—	0.7
Top-p	—	0.9
Maximum new tokens	256	256
Maximum high-level calls	5	5
Thinking mode	Disabled	Disabled
Base generation seeds	181	181, 193, 211, 223
Replay seeds	17, 29, 43	17, 29, 43

Agarwal et al., 2024). Successful-experience reuse also appears in self-imitation learning (Oh et al., 2018). STaR, ReST, and Re-ReST obtain new training examples from generated reasoning or trajectories, using correctness, reward, or reflection to select or improve them (Zelikman et al., 2022; Gulcehre et al., 2023; Dou et al., 2024). CCL fixes the recovery state and original obligation, then accepts an alternative through replay and semantic review. Its likelihood target is the accepted program rather than a teacher token distribution.

E.2 Action alignment and candidate selection

Action comparison normalizes JSON key order and whitespace while retaining the actor, skill, and arguments. The first unequal action or stop/continue boundary defines divergence; identical programs are excluded. Eligible alternatives are ordered by longest shared repair steps, smallest action edit distance, and shortest target program, with sampling index and program identifier breaking ties. The selected alternative then undergoes semantic review, as summarized in Algorithm E.1.

Algorithm E.1 Collection and learning from verified repair programs

Require: Training records $(x, c, \mathcal{C})$; frozen policy $\pi_{\theta_{\mathrm{SFT}}}$; replay seeds $\mathcal{Z}$
Initialize $\mathcal{D}_{\mathrm{CF}} \leftarrow \varnothing$ and success pool $\mathcal{P} \leftarrow \varnothing$
for each training record $(x, c, \mathcal{C})$ **do**
  Generate direct repair y^- and four alternatives from the same public x
  Freeze program identities and evaluate the fixed outputs
  if y^- is schema-valid and $R(c, y^-) = 3$ **then**
    Add its verified direct repair to $\mathcal{P}$
  else if y^- is schema-valid and $R(c, y^-) < 3$ **then**
    Form eligible alternatives using Equations 1 and 2
    if at least one alternative is eligible **then**
      Select y^+ with the fixed proximity rule
      if semantic review accepts the selected pair **then**
        Add (x, y^-, y^+) to $\mathcal{D}_{\mathrm{CF}}$
      end if
    end if
  end if
end for
Form an equally sized, semantically reviewed preservation set from $\mathcal{P}$
Retain disjoint correction and preservation contexts; freeze both sets
Initialize at θ_{SFT} and optimize Equation 5

In Algorithm E.1, x is the public input, c the private replay record, and $\mathcal{C}$ its fixed recovery contract. $\pi_{\theta_{\mathrm{SFT}}}$ is the repair model with frozen Repair-SFT parameters, and $\mathcal{Z}$ is the set of replay seeds. The source program is y^- and the selected alternative is y^+; $R(c, y)$ counts successful replays of program y. The set $\mathcal{D}_{\mathrm{CF}}$ contains accepted correction triples, while $\mathcal{P}$ is the pool of successful source programs available for preservation. The symbols $\varnothing$ and $\leftarrow$ mean the empty set and assignment, respectively.

E.3 INDEPENDENT SUCCESS PRESERVATION

Preservation targets are verified direct repairs from the same frozen model on training states disjoint from the correction set. Each must satisfy the fixed recovery contract for its state, although its actions may differ from the construction witness. We select equally sized correction and preservation sets, applying full-response supervision only to the latter.

Collection sizes. Each backbone constructs its own correction and preservation collections from the same 457 training states using its frozen Repair-SFT model. Table E.2 reports the final collection sizes after replay verification, divergence-boundary state matching, candidate selection, and semantic review. Depending on the backbone, this procedure yields 53–68 accepted counterfactual correction examples. Following the 1:1 preservation design above, we select an equal number of verified preservation examples from disjoint training states.

Table E.2: CCL training collection sizes for the four backbone experiments. N_{CF} is the number of accepted counterfactual correction examples after replay verification, divergence-boundary state matching, candidate selection, and semantic review. N_{pres} is the number of disjoint verified preservation examples selected at a 1:1 ratio.

Backbone	N_{CF}	N_{pres}	Total update examples
Qwen2.5-7B	57	57	114
Llama3.1-8B	53	53	106
Qwen3-8B	62	62	124
Qwen3.5-9B	68	68	136

This reuse of successful behavior relates to experience replay (Rolnick et al., 2019), while RL's Razor studies how update rules affect forgetting (Shenfeld et al., 2026). We accordingly evaluate retained successes separately from new recoveries.

F REGIONAL OBJECTIVE AND TOKEN ALIGNMENT

F.1 SUPERVISION COMPONENTS

Equation 3 combines the supervision components in Table F.1. The shared repair steps remains prediction context without direct correction labels; the divergent action, its field values, the remaining completion, and the native ending token receive their respective losses.

Table F.1: **Components of the implemented CCL objective.** Coefficients multiply independently averaged losses. A token can belong to more than one component. The preservation term uses a different set of contexts.

Component	Supervised positions	Coefficient
Divergence	Complete first changed action or target stopping boundary	1
Semantic fields	Actor, skill, and argument values of the divergent action	1
Completion	Accepted continuation after the divergent action	0.25
Ending	Model-native response ending token	0.5
Preservation	Full verified response from an independent context	0.5

For correction example $e = (x, y^-, y^+)$, x is the public input, y^- the source, and y^+ the accepted alternative. Write its target tokens as $u^+ = (u_1^+, \ldots, u_M^+)$, where M is the response length, t the token position, and $u_{<t}^+$ the preceding text. The model π_θ, with trainable parameters θ, gives token loss $\ell_t(\theta; e) = -\log \pi_\theta(u_t^+ \mid x, u_{<t}^+)$, using the natural logarithm. For mask $\mathcal{T}$, the set of selected token positions with size $|\mathcal{T}|$, its mean loss is

$$\bar{\ell}_{\mathcal{T}}(\theta; e) = \begin{cases} |\mathcal{T}|^{-1} \sum_{t \in \mathcal{T}} \ell_t(\theta; e), & |\mathcal{T}| > 0, \\ 0, & |\mathcal{T}| = 0. \end{cases} \tag{F.1}$$

The bar denotes the average over positions $t \in \mathcal{T}$. In Equation 3, masks D, S, C, and E select divergence, semantic field values, completion, and the native ending token. S selects the values of `agent`, `skill`, and `args`, not the keys; it overlaps D, and boundary or ending tokens may enter several masks. Each component is averaged independently before weighting by its coefficient, with $\lambda_c = 0.25$ for completion, $\lambda_e = 0.5$ for ending, and $\lambda_p = 0.5$ for the preservation term in Equation 5. Intentionally empty components contribute zero; a nonempty region with no mapped tokens is an error.

F.2 Token mapping and stopping boundaries

We map canonical action character spans through the chat template and tokenizer of each backbone, including a token in a mask when their character intervals intersect. Prompt and padding positions are excluded. The causal shift aligns target position i in the rendered sequence with the prediction logits at position $i - 1$.

If the source stops first, the first additional target action defines divergence; if the target stops first, its ending decision does. A target with no later actions has no completion component, but retains the ending-token loss. Sequences exceeding 4,096 tokens are rejected rather than truncated so that training preserves the entire accepted program.

F.3 Sample normalization and gradient accumulation

Let $N_{\mathrm{CF}} = |\mathcal{D}_{\mathrm{CF}}|$ and $N_{\mathrm{pres}} = |\mathcal{D}_{\mathrm{pres}}|$ count correction and preservation examples. For $s = (x_s, u_s)$, x_s is the public input and u_s its verified response of N_s tokens; N_s is a response length, not a dataset size. The loss $\mathcal{L}_{\mathrm{pres}}(\theta; s)$ averages the negative log likelihood of each token $u_{s,t}$ conditioned on x_s and preceding tokens $u_{s,<t}$.

A correction example inherits effective state weight b_e and uses $\alpha_e = (3 - R(c, y^-))/3$, the fraction of unsuccessful source replays from record c. Preservation instead uses $\tilde{b}_s = 1/|g_{\mathrm{pres}}(s)|$, where $g_{\mathrm{pres}}(s)$ groups selected preservation examples from the same episode and failure category. We normalize the two streams separately:

$$\omega_e = \frac{\alpha_e b_e}{N_{\mathrm{CF}}^{-1} \sum_{e' \in \mathcal{D}_{\mathrm{CF}}} \alpha_{e'} b_{e'}}, \qquad \nu_s = \frac{\tilde{b}_s}{N_{\mathrm{pres}}^{-1} \sum_{s' \in \mathcal{D}_{\mathrm{pres}}} \tilde{b}_{s'}}. \tag{F.2}$$

The weights ω_e and ν_s therefore have mean one within their respective sets; e' and s' index the examples summed in each denominator. The inherited b_e already incorporates the initialization group weight, while $\tilde{b}_s$ is recomputed on the preservation subset. In Equation 5, $\mathcal{L}_{\mathrm{CF}}$ sums the four regional losses and $\mathcal{L}_{\mathrm{CCL}}$ combines weighted correction and preservation means.

Each micro-iteration processes one example from each stream and backpropagates their losses separately, dividing both by the accumulation factor of eight, including the final partial group. Training uses one device and likelihood losses on accepted targets, in contrast to the direct comparison of preferred and rejected responses in DPO (Rafailov et al., 2023).

G Training and Repair Inference

G.1 Backbones and supervised initialization

The local backbones are `Qwen/Qwen2.5-7B-Instruct`, `meta-llama/Llama-3.1-8B-Instruct`, `Qwen/Qwen3-8B`, and `Qwen/Qwen3.5-9B`. Model reports and cards describe the corresponding published checkpoints (Yang et al., 2025b; Grattafiori et al., 2024; Meta AI, 2024; Yang et al., 2025a; Team, 2025; Qwen Team, 2026). The original-model rows in Table 1 use these published models before project-specific repair adaptation. Inputs to the repair model are textual, including for the Qwen3.5 architecture.

Repair-SFT uses the 457 verified training witnesses. Prompt tokens are masked, and the assistant response and native ending token provide the supervised target. Every witness is presented once per epoch with the episode–mechanism weight. LoRA (Hu et al., 2021) provides the trainable adapters

in BF16 without weight quantization. The attention projections and feed-forward projections are adapted according to the module structure of each backbone.

G.2 Optimization settings

Table G.1 records the shared training settings. The Repair-SFT checkpoint used for collection is fixed at epoch two. Its cosine schedule follows the original six-epoch plan, with the checkpoint taken after 116 updates. The CCL stage starts from the corresponding frozen Repair-SFT checkpoint and runs for six epochs on the accepted correction and preservation collections. Candidate generation is completed before this update.

Table G.1: **Shared optimization settings.** Maximum length applies to the complete rendered training sequence; the generation budget is separate.

Setting	Repair-SFT	CCL update
Training precision	BF16	BF16
Weight quantization	None	None
LoRA rank / scaling	32 / 64	32 / 64
LoRA dropout	0.05	0.05
Optimizer	AdamW	AdamW
Learning rate	5×10^{-5}	5×10^{-6}
Weight decay	0	0
Learning-rate schedule	Cosine	Cosine
Warmup updates	17	1
Microbatch size	1	1
Gradient accumulation	8	8
Maximum sequence length	4,096	4,096
Training seed	181	181

The initialization schedule is configured for 348 updates; CCL uses a separate schedule over its fixed accepted data. For the scope controls S1, S3, S4, and S5, a fixed set of correction pairs and an equal number of preservation examples is used for six epochs (48 updates) under training seeds 181, 193, and 211. The coefficient study reuses this collection and training protocol; replay seeds remain distinct from training seeds.

G.3 Generation, parsing, and execution

Direct evaluation uses greedy decoding with thinking disabled, at most 256 new tokens, and a five-call program limit. The native model ending condition or balanced-JSON stopping terminates generation. Tokens are generated without grammar-constrained decoding. Parsing and compilation then determine whether the original output is a legal program. Malformed outputs remain failures; the evaluation does not repair their JSON or replace them with a newly sampled candidate.

A program that passes these interface checks is submitted to the skill driver in its fixed order. The local policy is invoked once per repair context. Simulator search and candidate verification occur during offline collection, whereas evaluation replays the already generated program. Model calls made by a complete cooperative controller are accounted for separately from the single generation call of the local policy.

H Evaluation Protocols and Baseline Adaptations

H.1 Local recovery and program validity

All methods in Table 2 follow the same local recovery evaluation rules: the Test183 state set, public input fields, five-call program limit, checkpoint replay, and strict-recovery predicate. Each returned program is fixed before replay under 3 seeds. Let $\mathcal{S}$ be the set of test-state indices, with $|\mathcal{S}| = 183$. For index $i \in \mathcal{S}$, y_i is the generated repair program, c_i its private replay record, and $\mathcal{C}_i$ the fixed recovery contract. The seed set is $\mathcal{Z} = \{17, 29, 43\}$ and z denotes one of its elements. The verifier

$V(c_i, y_i, z; \mathcal{C}_i) \in \{0, 1\}$ reports whether that replay passes the recovery checks. Define

$$r_i = \prod_{z \in \mathcal{Z}} V(c_i, y_i, z; \mathcal{C}_i), \qquad \text{StrictRecovery} = \frac{100}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} r_i. \tag{H.1}$$

The product makes r_i a binary strict-recovery indicator: it is one only when every replay succeeds. The sum counts successful test states, and the factor $100/|\mathcal{S}|$ converts that count to a percentage. Invalid programs have $r_i = 0$ and remain in the denominator. Family scores apply the same rule to the corresponding subsets, with sizes shown in Table B.1.

Program validity is the fraction of outputs accepted by the parser and skill interface. It is computed before asking whether the task obligation was restored. The call metric counts model invocations used to obtain the final program, including intermediate feedback and refinement calls. Automatic replay of that program does not add a generation call. Paired recovery statistics are defined in Section I.

H.2 Local adaptations of repair methods

The compared methods use the public context and return at most five skill calls. Their local adaptations preserve the mechanism of each approach while expressing its final answer in the project JSON interface. We describe these adaptations here using the suffix Local; the main table uses the short method names.

Public Heuristic and prompted programs. Public Heuristic selects actions through deterministic rules over public failure feedback and state. It uses no language model. ProgPrompt-Local supplies the available skills and structured program examples to a Qwen3-8B generator (Singh et al., 2022). The output is a JSON skill program rather than executable Python. Its generation uses one model call.

Iterative refinement. Self-Refine-Local follows an initial draft with self-feedback and revision (Madaan et al., 2023). CMFR-Local implements a local multi-stage feedback and revision procedure (Farag et al., 2025). Both use three model calls with Qwen3-8B. Their additional analysis stages allow up to 192 new tokens per call, followed by the 256-token final program. Intermediate reasoning remains separate from the executed JSON response.

Learning from critical steps. CSO-Local retains expert action proposals, a process reward model, student continuation, and simulator validation (Li et al., 2026a). Accepted critical steps train a reference-based DPO objective (Rafailov et al., 2023) with $\beta = 0.5$, where β scales the preference logit formed from policy-to-reference log-probability ratios. The frozen Repair-SFT model is the reference. Deployment generates one repair program in a single local model call.

Learning from reflective trajectories. Agent-R-Local retains failed-trajectory revision, branch consistency checks, and a mixture of successful and revised trajectories (Yuan et al., 2025). Failed response portions are masked during supervision. Like CSO-Local, it starts from a Qwen3-8B Repair-SFT checkpoint and uses one local generation call at evaluation.

All final programs use the evaluation in Section H.1. The call column in Table 2 measures inference invocations.

H.3 TDW-MAT baselines

TDW-MAT evaluates cooperative object transport in ThreeDWorld (Gan et al., 2020; Zhang et al., 2024). The test set contains 24 episodes, evenly divided between Food and Stuff, with a common budget of 3,000 simulator frames. The recorded RHP, REVECA, PCE, and CoELA baseline runs use oracle perception with TDW Python 1.11.23.5 and Unity 1.11.23. Their language-model planning calls use `gpt-4o-2024-11-20` with temperature 0.7, top-p 1, and a maximum of 1,024 output tokens. RHP uses rules instead of a language model.

Food and Stuff scores average terminal transport rates within their episode sets; Overall averages all test episodes with equal episode weight. The main table reports these results together with TR-AUC under the common frame budget.

Table H.1: **TDW-MAT baseline controllers.** The methods retain their respective coordination and planning mechanisms.

Baseline	Controller design
RHP + RHP (Zhang et al., 2024)	Two rule-based controllers provide the non-LLM reference.
CoELA (Zhang et al., 2024)	A modular controller connects communication with language-model planning and skill execution.
REVECA (Seo et al., 2025)	Adaptive planning uses information relevance and relative proximity, with trajectory-based validation.
CaPo (Liu et al., 2025)	Cooperative plan optimization adapts execution to shared task progress.
PCE (Seo et al., 2026)	Uncertainty-aware planning uses the evolving execution context to revise decisions.

Repair models. The SFT and CCL repair modules integrated into CoELA and PCE use Qwen3.5-9B. The CCL update uses repair examples from the TDW-MAT training split. This applies the CCL training procedure to TDW-MAT before integrating the resulting repair model into the cooperative controllers.

I Supervision Controls and Additional Analysis

I.1 Supervision scope

Table I.1 separates ordinary full-response training from the supervision variants used in CCL. The Full-response CE control applies ordinary cross entropy to complete successful repair responses, without additional divergence, semantic-field, or ending losses. S1 instead uses the full response for its primary correction term while retaining the shared auxiliary terms. S2 supervises the full suffix, including the divergent action; completion begins after it. S3 supervises all modified target actions. S4 and S5 emphasize the first divergent action with completion coefficients $\lambda_c = 0$ and 0.25, respectively. S1, S3, S4, and S5 use the data and optimization protocol described in Section G.2. S0 provides the Qwen3-8B Repair-SFT test reference from the main evaluation. Figure I.1 plots the reported test recovery values.

Table I.1: **Supervision controls.** Recovery and validity are percentages under unconstrained generation. Training variants report mean scores; S0 uses the main-paper Repair-SFT test reference. An em dash denotes an unreported validation result.

	Strict recovery (%)		Program validity (%)
Condition	Val116	Test183	Test183
S0: Repair-SFT	—	39.89	97.81
Full-response CE	57.47	43.90	95.99
S1: full response + auxiliary terms	59.20	44.99	94.35
S2: full suffix	56.03	40.44	91.99
S3: modified actions	57.47	46.27	93.08
S4: first divergence, $\lambda_c = 0$	55.74	46.27	92.35
S5: CCL, $\lambda_c = 0.25$	58.91	**47.54**	92.71

CCL achieves 47.54% test recovery, 3.64 percentage points above the added Full-response CE control. It also exceeds S1 at 44.99% and S3/S4 at 46.27%, the variants retaining auxiliary supervision.

I.2 Completion coefficient

We vary λ_c on Val116 under the data and training protocol of the scope study, keeping the preservation ratio at 1:1 and its coefficient at 0.5. Mean recovery is highest at $\lambda_c = 0.25$; both smaller and larger coefficients yield lower means (Figure I.2).

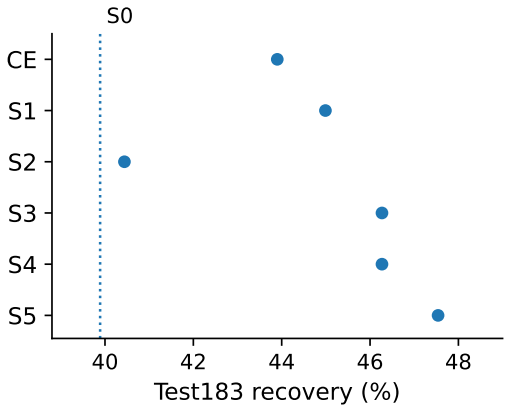


Figure I.1: **Supervision controls.** Test recovery for CE (Full-response CE) and S1–S5 in Table I.1; the dotted line denotes S0.

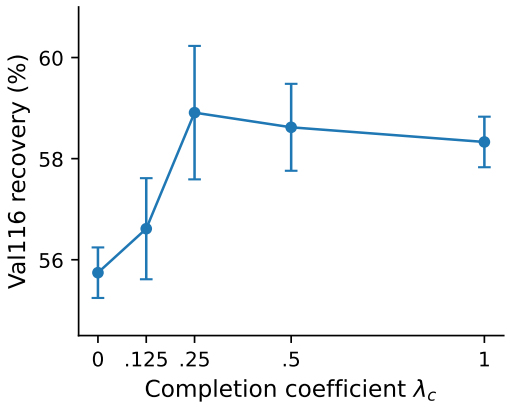


Figure I.2: **Completion strength.** Val116 mean recovery and sample standard deviation over three training seeds for each tested coefficient.

I.3 Paired recovery outcomes

For test-state index $i \in \mathcal{S}$, let r_i^{SFT} and r_i^{CCL} indicate strict success of the fixed programs from the two models: one if all replays succeed and zero otherwise. The paired counts are

$$n_{\mathrm{keep}} = \sum_{i \in \mathcal{S}} r_i^{\mathrm{SFT}} r_i^{\mathrm{CCL}}, \qquad n_{\mathrm{gain}} = \sum_{i \in \mathcal{S}} (1 - r_i^{\mathrm{SFT}}) r_i^{\mathrm{CCL}}, \tag{I.1}$$

$$n_{\mathrm{lost}} = \sum_{i \in \mathcal{S}} r_i^{\mathrm{SFT}} (1 - r_i^{\mathrm{CCL}}), \qquad n_{\mathrm{fail}} = \sum_{i \in \mathcal{S}} (1 - r_i^{\mathrm{SFT}})(1 - r_i^{\mathrm{CCL}}). \tag{I.2}$$

Here n_{keep} counts states solved by both models, n_{gain} those solved only after CCL, n_{lost} those solved only before CCL, and n_{fail} those solved by neither. The sums over i partition $|\mathcal{S}|$ test states, giving net recovery change and retention:

$$\Delta\mathrm{StrictRecovery} = \frac{100(n_{\mathrm{gain}} - n_{\mathrm{lost}})}{|\mathcal{S}|}, \qquad \mathrm{Retention} = \frac{n_{\mathrm{keep}}}{n_{\mathrm{keep}} + n_{\mathrm{lost}}}. \tag{I.3}$$

Δ denotes the CCL rate minus the Repair-SFT rate in percentage points. Retention is the fraction of initial successes retained, defined when $n_{\mathrm{keep}} + n_{\mathrm{lost}} > 0$; multiplying by 100 expresses it as a percentage. Figure 7 reports these outcomes separately from validity, while the coefficient study reports variation across training runs.

I.4 Collection and deployment workload

The full dataset construction and validation process involved more than 30,000 simulator replays. Within that process, collecting one direct repair and four alternatives for each of the 457 training states defines 2,285 programs per backbone and 27,420 program–seed keys across four backbones. These collection keys include invalid outputs that do not complete a physical repair; they are distinct from the full construction workload. Fixed-program replay makes no language-model call. Deployment generates one bounded repair per context; Table 2 separately reports extra calls used by prompting-based refiners.

J Qualitative Records and Executed Repairs

J.1 Representative PARTNR-FS records

The four records below retain their frozen training membership and primary failure categories. Each panel separates public input from the private local obligation and verified construction witness. Actor 0 is Alice and actor 1 is Bob. Actions use compact actor: skill[arguments] notation; the original JSON and complete public messages accompany the source package. Actor capabilities are specified in Table C.2. The saved witness audits report successful restoration, usable task history, local responsibility closure, and no task-state regression for all four records.

D1: Reachability: approach before placing

Record. Train; episode 1223.
Task. Help me exchange these objects. First, place the butter dish on kitchen counter. Then, move the butter dish to the table in the bedroom. After that, place the egg on the dining room table. Finally, move the egg to the chair in the kitchen.
Failed call. `Place egg_1, on, table_29, None, None.`
Feedback. Unexpected failure! - Failed to place! Not close enough to table_29 or occluded.
Public state and recent context. Bob has picked up the egg, while Alice is still exploring the kitchen. The placement feedback reports insufficient proximity or occlusion at the dining table.

Private local obligation. place held egg on dining table before kitchen-chair transfer.
Verified construction witness.

```
Agent 1: Navigate[table_29]
Agent 1: Place[egg_1, on, table_29]
```

Recorded endpoint. egg_1 on table_29. The witness completes the current dining-table stage. The later transfer to the kitchen chair remains downstream.

D2: Holding: a placement prerequisite across agents

Record. Train; episode 1425.
Task. Move the toy airplane and helmet from the table in the living room to the shelves in the bedroom. Place them next to each other on the shelves.
Failed call. `Rearrange toy_airplane_0, on, shelves_7, next_to, helmet_1.`
Feedback. Unexpected failure! - No valid placements found for entity shelves_7. It looks like the spatial constraint next_to is not feasible because the reference object helmet_1 either does not exist or has not yet been placed on the shelves_7. Try placing the grasped object toy_airplane_0 on shelves_7 without the spatial constraint next_to.
Public state and recent context. Alice holds the toy airplane and Bob holds the helmet. Their recent history contains two rearrangement requests, but the helmet is not yet on the destination shelves.

Private local obligation. place held airplane and helmet together on bedroom shelves.
Verified construction witness.

```
Agent 1: Navigate[shelves_7]
Agent 1: Place[helmet_1, on, shelves_7]
Agent 0: Navigate[shelves_7]
Agent 0: Place[toy_airplane_0, on, shelves_7, next_to, helmet_1]
```

Recorded endpoint. helmet_1 on shelves_7; toy_airplane_0 on shelves_7. This witness places the helmet before the airplane, establishing the reference object for the relative placement. The learned rollout in Section J.3 uses the reverse object order.

D3: Unsupported action: bypass a non-articulated container

Record. Train; episode 27.
Task. Bring the toy vehicles and toy food from the bedroom to the table in the living room.
Failed call. `Open chest_of_drawers_37`.
Feedback. Unexpected failure! - Failed to Open! chest_of_drawers_37 is not articulated - and cannot be Opened.
Public state and recent context. Both agents have explored the bedrooms. The public object records locate the toy food at the chest of drawers even though the requested opening operation is unavailable for that furniture.

Private local obligation. move one exposed task food from non-articulated chest to a valid living-room table.
Verified construction witness.

```
Agent 1: Navigate[chest_of_drawers_37]
Agent 1: Pick[toy_food_2]
Agent 1: Navigate[table_18]
Agent 1: Place[toy_food_2, on, table_18]
```

Recorded endpoint. toy_food_2 on table_18. The contract admits several object–destination branches. This four-call witness completes the toy-food-2/table-18 branch; the remaining object transfers belong to the broader task.

D4: Relation: use a supported placement argument

Record. Train; episode 675.
Task. Help me move the hand towel and soap dish from the bathroom cabinet to the laundry room cabinet.
Failed call. `Place soap_dish_1, "on", cabinet_56, None, None`.
Feedback. Unexpected failure! - No valid placements found for entity due to spatial relation can only be 'on' and 'within'
Public state and recent context. Bob already holds the soap dish at the laundry-room cabinet. Recent placement attempts contain quoted relation values and attempts to use within.

Private local obligation. place held soap dish on laundry-room cabinet 56.
Verified construction witness.

```
Agent 1: Place[soap_dish_1, on, cabinet_56]
```

Recorded endpoint. soap_dish_1 on cabinet_56. The canonical on relation produces the required cabinet-top placement. The hand towel remains a separate task responsibility.

J.2 VERIFIED REPAIR-PROGRAM PAIRS

C1–C3 are training pairs collected from the same frozen Qwen3-8B Repair-SFT model. Each source is its direct repair and each alternative is a sample from that same model. All three pairs share one action before diverging. The records establish within-seed state agreement at that boundary and local recovery of 0/3 versus 3/3. The complete programs remain the units of verification.

C1: A missing reference object

Record. Train; episode 1418.
Task and failure context. Alice holds the teapot while the pitcher remains on the original table. The current stage is to place them next to one another on another living-room table before transferring them to the bedroom.
Direct repair.

```
Agent 0: Navigate[table_25]
Agent 0: Place[teapot_0, on, table_25, next_to, pitcher_1]
```

Verified alternative from the same model.

```
Agent 0: Navigate[table_25]
Agent 0: Place[teapot_0, on, table_25]
Agent 0: Rearrange[pitcher_1, on, table_25, next_to, teapot_0]
```

Learning regions. After a shared navigation, the source places the teapot relative to an absent pitcher. The alternative first places the teapot, then brings the pitcher beside it. The second action defines the primary and semantic-field masks; the third supplies completion supervision.

C2: A relation-field correction

Record. Train; episode 675.
Task and failure context. Bob is transporting the soap dish to the laundry-room cabinet. The source navigates successfully but requests a within placement on furniture without a corresponding receptacle.
Direct repair.

```
Agent 1: Navigate[cabinet_56]
Agent 1: Place[soap_dish_1, within, cabinet_56]
```

Verified alternative from the same model.

```
Agent 1: Navigate[cabinet_56]
Agent 1: Place[soap_dish_1, on, cabinet_56]
```

Learning regions. Both programs contain two actions and differ only in the relation argument of the second action. The accepted on placement closes the soap-dish obligation. No later action supplies completion supervision. This is a different context from D4 despite the shared episode number.

C3: Continuing after a successful but incomplete action

Record. Train; episode 620.
Task and failure context. The instruction requires moving a shoe and a milk-frother cup to the laundry-room shelves. The original recorded failure concerns reaching the cup; the frozen failure family remains Reachability.
Direct repair.

```
Agent 0: Rearrange[shoe_0, on, shelves_20]
```

Verified alternative from the same model.

```
Agent 0: Rearrange[shoe_0, on, shelves_20]
Agent 1: Rearrange[milk_frother_cup_1, on, shelves_20]
```

Learning regions. The source moves the shoe successfully and stops while the cup obligation remains open. The alternative retains that action and assigns the cup transfer to Bob. The first divergence is therefore a stop/continue boundary, not the first physically failed action. The second target action is supervised together with its semantic fields and the accepted program ending.

Table J.1: **Checks on the three training pairs.** Recovery uses the full program under seeds 17/29/43. The last column reports the original source replay field verbatim; it is true in the placement-failure cases C1/C2, whose shared navigation nevertheless executes.

Pair	Episode	Source	Target	Shared repair steps match	Source `invalid_program`
C1	1418	0/3	3/3	Yes	True
C2	675	0/3	3/3	Yes	True
C3	620	0/3	3/3	Yes	False

C1 includes reconstructed task history; C2 and C3 use checkpoint-state certification. Shared repair steps agreement is checked within each replay seed. The accompanying records retain the exact history-basis identifiers, program identities, digest checks, and original success flags for all 18 program–seed outcomes.

J.3 A REPAIR GENERATED BY THE TRAINED POLICY

C4 revisits training episode 1425 from the restored state shown in Figure 4. Alice holds the airplane and Bob holds the helmet. A Qwen3-8B CCL model produces the four-call program below through one unconstrained direct generation. The recorded demonstration then replays that fixed program under seed 17.

C4: Qwen3-8B CCL, generated repair and recorded seed-17 replay

Generated repair.

```
Agent 0: Navigate[shelves_7]
Agent 0: Place[toy_airplane_0, on, shelves_7]
Agent 1: Navigate[shelves_7]
Agent 1: Place[helmet_1, on, shelves_7, next_to, toy_airplane_0]
```

Recorded result. All four skill calls succeed; the local obligation closes and both objects are released on the shelves. The program places the airplane first, unlike the helmet-first construction witness D2.

The archived first GPT-4o local decision places the airplane but then requests `Rearrange` on the helmet that Bob already holds. The latter call returns "This object is already held by this agent." Its recorded local decision fails, whereas the generated CCL program uses a direct placement for the held helmet. Later GPT-4o decisions recover in the original episode. The comparison therefore concerns the recorded local decisions, with C4 serving as a training-state demonstration of the learned policy.

The supplementary videos show these recorded local decisions with viewing pauses. They illustrate this training-state repair rather than providing timing measurements or additional test trials.