Beyond Episodes: Learning Behaviors That No Rollout Demonstrated

Anonymous authors
Paper under double-blind review

Abstract

An LLM agent's rollouts can contain complementary fragments that together solve a task, even when no individual rollout succeeds. Single-episode supervision leaves these combinations undemonstrated. We investigate whether agents can learn these behaviors by composing their own fragmented interaction experience. We propose **GraphStitch**, which constructs goal-conditioned training trajectories by joining fragments from different rollouts at shared states. By pooling both successful and failed attempts, it recovers solutions absent from every original rollout. Specifically, GraphStitch organizes observed transitions into a shared graph, extracts composed paths, and assigns goals using environment-specific achievement rules. State identity or environment replay checks executability before ordinary supervised fine-tuning. This extends supervision beyond episode boundaries while reusing recorded action-labeled edges as its building blocks. We evaluate compositional goal reaching and generalization to held-out tasks. On held-out Blocksworld, GraphStitch improves greedy success over hindsight relabeling by **28.2 percentage points**. Using one-quarter as many collected rollouts as Hindsight, GraphStitch also achieves higher mean held-out greedy success on Blocksworld (68.0% versus 46.8%), with additional replay during data construction. Gains also appear on Countdown, and controlled comparisons support the contribution of cross-episode composition. Composing fragmented experience thus enables LLM agents to learn beyond behaviors demonstrated in individual rollouts.

1 Introduction

LLM agents use reinforcement learning (RL) to learn from interactions with an environment and improve their ability to solve tasks (Wang et al., 2025c; Feng et al., 2025). Yet the steps needed for a solution may be scattered across different rollouts. Consider two rollouts from different initial states that pass through the same intermediate state. The first reaches this shared state but fails to complete its task; the second contains the remaining steps to that task's goal. Joining the first rollout's prefix with the second's suffix produces a complete solution absent from either original rollout (Figure 1). **Can an agent learn a behavior that never occurred in any of its original rollouts?**

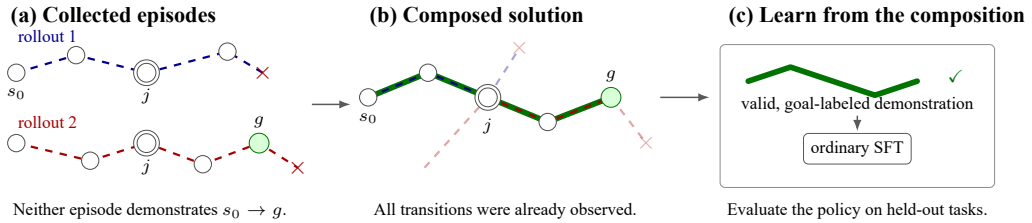


Figure 1: Beyond episode boundaries. (a) Independently collected rollouts contain complementary fragments; crosses mark failure on their original tasks. (b) Joining at the same underlying state j produces a solution to g absent from both episodes. (c) After validity and goal checks, the composition supplies supervision for the agent. Figure 3 shows a real instance.

Episode boundaries record how experience was collected; they need not determine its supervision. Success-filtered fine-tuning retains completed attempts (Yuan et al., 2023). Hindsight supervision relabels individual episodes with achieved goals (Ghosh et al., 2021; Li et al., 2026b). Recovering

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.

additional supervision matters when sparse feedback makes successes difficult to collect (Huo et al., 2026). Hindsight asks what each episode achieved. We ask what the collection of episodes makes possible.

We propose **GraphStitch** to construct these missing demonstrations for LLM agents. It pools observed transitions into a graph, joins fragments at shared state keys, and pairs accepted paths with achieved goals. State identity or environment replay checks executability before ordinary supervised fine-tuning (SFT; §3); goal correctness also depends on the relabeling rules. The constructor reuses recorded state-key edges to form complete demonstrations. Figure 3 gives a real example from two failed rollouts.

This approach builds on trajectory stitching as combinatorial generalization in RL (Ghugare et al., 2024) and earlier methods that construct stitched data for policy learning (Char et al., 2022; Hepburn & Montana, 2022). LLM-agent trajectory recombination has also been studied (Lin et al., 2025; Xu et al., 2026). We study the learning value of state-based compositions from agents' own fixed experience; §2 distinguishes these approaches.

We test whether pools contain undemonstrated solutions and whether training on them improves policies. For 98 of the 301 Blocksworld collection tasks, GraphStitch recovers a solution although no single source episode provides one. Learning gains extend to held-out Blocksworld and Countdown tasks; partition and budget controls support the role of composition. In the Blocksworld budget comparison, composing the existing rollout pool yields higher held-out success than giving Hindsight four times as many collected rollouts (§4.5). A behavioral probe tests goals requiring composition in its source pool. ALFWorld exposes a boundary: most candidates fail replay, and surviving data yield no detectable gain over hindsight (§4).

Our contributions are:

- **Supervision beyond episode boundaries.** GraphStitch constructs goal-labeled demonstrations by composing observed transitions across agent rollouts, then trains on the accepted paths (§3).
- **Evidence for learning from undemonstrated solutions.** We establish that such solutions exist in the collection pool and that training on them improves held-out behavior, with controls for composition and data volume (§4.2–4.5).
- **A test of compositional goal reaching.** Increment pairs and a composition-necessity check distinguish goals reached within episodes from those supported by their combination (§3.7, §4.6).

2 Related Work

2.1 Learning combinations absent from source trajectories

Ghugare et al. (2024) frame stitching as combinatorial generalization and augment imitation with goals sampled from other trajectories through nearby states. GraphStitch shares the data-augmentation principle, but constructs complete goal-labeled paths for LLM-agent SFT, rather than augmenting individual state–action–goal targets.

Stitching followed by policy learning also predates this work. BATS (Char et al., 2022) and model-based trajectory stitching (Hepburn & Montana, 2022) use learned dynamics to add connections and train policies from the resulting trajectories; DiffStitch (Li et al., 2024) generates connecting transitions with diffusion. State-level stitching improves imitation from imperfect demonstrations (Wang et al., 2025b). GraphStitch studies this principle with agents' own language-action rollouts, restricting graph search to recorded action-labeled edges between shared state keys.

Stitching in learned policies depends on the algorithm and coverage: return-conditioned learning, including Decision Transformer (Chen et al., 2021), relies on coverage assumptions (Brandfonbrener et al., 2022), and sufficiently expressive critics can enable Monte-Carlo stitching (Bortkiewicz et al., 2025). Q-learning Decision Transformer (Yamagata et al., 2022) changes return targets, while self-predictive representations (Lawson et al., 2025) change the representation objective. Our intervention changes the demonstrations supplied to ordinary SFT.

2.2 Reusing experience in LLM-agent post-training

GCSL (Ghosh et al., 2021), HER (Andrychowicz et al., 2017), hindsight supervised learning (Li et al., 2026b), and AgentHER (Ding, 2026) reuse episodes through achieved-goal relabeling. EEF (Lan et al., 2025) extracts actions from expert failures, and ETO (Song et al., 2024) learns trajectory preferences. Single-episode relabeling changes what a recorded behavior is a demonstration of. GraphStitch additionally constructs behaviors absent from the source episodes. Our shared-pool Hindsight comparison tests the value of these additions beyond selected single-episode relabels. This controlled baseline does not reproduce the full HSL pipeline, which additionally uses multiple achieved goals, action masking, and reweighting.

SALT (Li et al., 2026a), GiGPO (Feng et al., 2025), and G2PO (Wang et al., 2026a) use trajectory graphs or repeated states for advantage estimation; GraphPO (Zhan et al., 2026) also shares suffixes during rollout generation. ILQL (Snell et al., 2022) and ArCHer (Zhou et al., 2024) use value-based language-policy learning. GraphStitch uses the graph for post-collection demonstration construction, leaving the SFT optimizer unchanged. It tests data composition rather than a new credit-assignment algorithm.

2.3 Trajectory recombination, search, and verification

SE-Agent (Lin et al., 2025) uses LLM-guided revision, recombination, and refinement to improve candidate trajectories for the same task. Both methods exploit cross-trajectory information. Its inference-time trajectory rewriting differs from our state-key joins and supervised post-training: we evaluate learned policies on held-out tasks without searching the experience graph. BES (Xu et al., 2026) also supports post-training, combining candidate evolution with backward subgoal decomposition. Our constructor instead extracts paths from a fixed transition pool, without model-generated expansion.

Other methods generate training experience by different means: STeCa (Wang et al., 2025a) calibrates suboptimal steps of the agent's own trajectories through reflection; TOFFEE (Wang et al., 2026b) combines MCTS with cross-task prefix reuse; DreamGym (Chen et al., 2025) synthesizes transitions with an experience model. State2State (Lei et al., 2026) turns explored states into new state-reaching tasks. GraphStitch constructs full demonstrations from recorded state-key edges and assigns achieved-goal labels, using replay to check executability under aliased keys.

Graphs also support planning in SoRB (Eysenbach et al., 2019) and GAS (Baek et al., 2025), while Go-Explore (Ecoffet et al., 2019) and Intelligent Go-Explore (Lu et al., 2024) use state archives for exploration. LATS (Zhou et al., 2023) searches at inference time; Agent Q (Putta et al., 2024) and ExACT (Yu et al., 2024) learn from search. Our empirical question concerns the training value of compositions recoverable from fixed experience, beyond single-episode reuse.

3 Method

3.1 Episodic experience and composed behavior

A deterministic environment $E = (S, A, T, \Omega, \omega)$ has states S, actions A, a partial transition function $T : S \times A \rightharpoonup S$ (undefined for inadmissible actions), and observations $\omega : S \to \Omega$. A task $t = (s_0^t, g^t)$ specifies an initial state and a goal predicate. Supervised conditions share a fixed rollout pool P. Its *compositional support* consists of executable paths assembled from recorded action-labeled state-key edges, including combinations absent from individual episodes. Here, undemonstrated means absent from P before construction and replay. GraphStitch selects paths for supervision. Collection, replay, and training budgets are reported separately (§4).

3.2 State keys: exact and aliased

Definition 1 (state keys)**.** *An* anchor $\kappa = \alpha \circ \omega : S \to K$ *maps observations to node keys. Two states are* bisimilar *if they admit the same actions, agree on every goal predicate, and matching actions lead to bisimilar successors. The key is* exact *if* κ *refines bisimilarity on the reachable states, and* aliased *if some key bucket merges non-bisimilar states.*

Joining episodes requires compatible intermediate states. Nodes are state-key buckets: exact keys preserve executability and goal predicates; aliased keys can join incompatible states (§4, A.1–A.4).

3.3 Constructing demonstrations across episode boundaries

Algorithm 1 gives the constructor and SFT procedure; §4.3 specifies the Countdown variant.

Algorithm 1 GraphStitch: learning from composed experience

Require: rollout pool P, key κ, deterministic environment E, goal rules, prefix selector $\mathcal{F}$, initial policy π_θ

$\mathcal{D}_{\text{hind}} \leftarrow$ successful rollouts $\cup$ eligible relabels from $\mathcal{F}(P)$; $\mathcal{C} \leftarrow \varnothing$
$G_P \leftarrow$ graph of executed transitions $\kappa(s) \xrightarrow{a} \kappa(s')$, retaining episode provenance
for each task $t = (s_0^t, g^t)$ **do**
  $\Sigma_t \leftarrow$ short graph paths from $\kappa(s_0^t)$ under fixed eligibility and search caps
  for each candidate $\sigma \in \Sigma_t$ **do**
    replay σ from s_0^t; reject inadmissible compositions
    assign an original or achieved goal g; retain the prefix ending at the selected achievement
    add $(t, g,$ replayed actions and text$)$ to $\mathcal{C}$ if accepted by the goal rule
  end for
end for
$\mathcal{D}_{\text{GS}} \leftarrow \mathcal{D}_{\text{hind}} \cup \mathcal{C}$, deduplicated by $(\text{task}, \text{goal}, \text{actions})$
return $\text{SFT}(\pi_\theta, \mathcal{D}_{\text{GS}})$ and $\mathcal{D}_{\text{GS}}$

Experience graph. Every executed transition in P contributes an action-labeled edge between state keys, retaining source-episode provenance. Paths can follow one episode to a shared key and continue along another. With complete-state keys, this composes observed state transitions. Under aliased keys, replay may produce a full-state transition absent from P even though its action-labeled key edge was recorded; replay legality does not establish full-state provenance.

Candidate paths. Breadth-first search constructs short paths from task initial keys to selected reachable endpoints or achievement edges. Endpoints need not be goals of successful rollouts. Eligibility, path counts, and depth caps are fixed per environment (§4, A.1–A.4).

Achieved goals. Retain walks up to original-goal success or an eligible achievement derived from executed states or actions; exclude walks without one. These selection rules also determine Hindsight's single-episode prefixes (A.1).

3.4 Executability: exact keys and the replay gate

A demonstration must execute from the task's initial state and satisfy its goal. Exact keys establish executability (Definition 1); with full-state rendering, replay is optional. Aliased-key candidates are replayed from the true initial state, rejecting inadmissible actions and using replayed text for training.

Goal validity additionally requires sound environment predicates or action-derived relabeling rules; A.4 describes the synthetic-goal label limitations in the ALFWorld datasets. One replay checks executability under determinism.

3.5 Extending supervision beyond episode boundaries

Won-only imitates successful rollouts. **Hindsight** adds selected achieved-goal prefixes from individual rollouts, following the goal-conditioned supervised learning approach (Ghosh et al., 2021; Li et al., 2026b). **GraphStitch** adds accepted graph paths to Hindsight. Deduplication on (task, goal, actions) removes repeated examples. Environment-specific selection rules are given in A.1–A.4.

Thus **Won-only** $\subseteq$ **Hindsight** $\subseteq$ **GraphStitch** for this union-based construction; the Countdown experiments use independently extracted sets (§4.3). Each condition uses ordinary SFT, so the intervention is which behaviors receive supervision. Our Hindsight baseline selects environment-specific prefixes; it is not a full HSL reproduction. A.6 gives a standard objective interpretation conditional on demonstration validity.

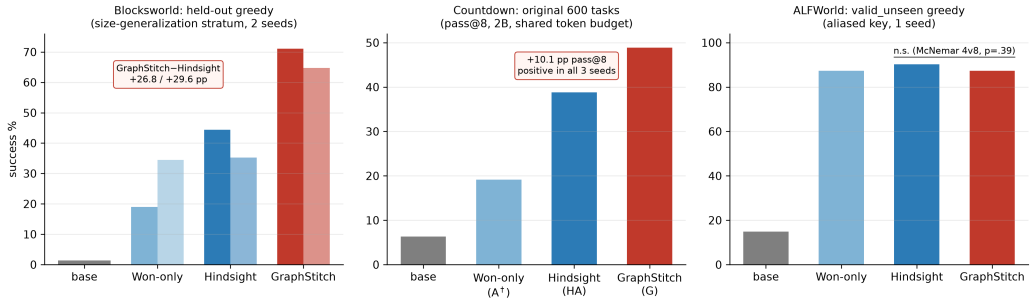


Figure 2: Policy performance across three environments. base → Won-only → Hindsight → GraphStitch under one collection budget per environment. Blocksworld: greedy, both seeds (solid and pale bars). Countdown: pass@8 on the original 600-task set with a 2B backbone, averaging all three seeds; 83 inputs overlap with G/HA relabeled training inputs (Table 6); A† includes the Won-only run with format collapse. ALFWorld: valid_unseen greedy; no GraphStitch-vs-Hindsight difference is detected in any of the four held-out cells (§4.4).

3.6 Training objective and trajectory representation

Supervised conditions share the backbone, optimizer recipe, and evaluation-compatible renderer within each environment. Assistant-token loss covers complete plans or individual actions according to the interface (A.1); §4.1 and §4.5 specify budgets and controls.

3.7 Testing goals supported by composition

The probe tests whether policies realize graph-supported reachability, comparing previously reached goals with goals supported by composition.

Definition 2 (increment pairs and the stitching score)**.** *Given episodes D in E, build G_D from D alone (§3.3). For task t,* $\mathrm{Own}_D(t)$ *contains keys visited by its own episodes;* $\mathrm{Reach}_D(t)$ *contains keys reachable from $\kappa(s_0^t)$ in G_D within the search cap. Define the* increment $\mathrm{Inc}_D(t) = \mathrm{Reach}_D(t) \setminus \mathrm{Own}_D(t)$. *An increment endpoint x is* composition-necessary *if no single episode of D visits $\kappa(s_0^t)$ and later x. An* increment pair $(t, g(x))$ *uses $x \in \mathrm{Inc}_D(t)$ and a goal $g(x)$ it satisfies; an* own pair *uses $x \in \mathrm{Own}_D(t)$. The* stitching score *divides the policy's increment-pair success rate by its own-pair success rate when the latter is nonzero. Different task indices can yield identical policy inputs (A.5).*

Some increment endpoints are reachable within a single foreign episode; the composition-necessity check separates these. A stricter post-hoc check excludes goals satisfied by *any state* such an episode reaches, and inputs matching supervised training prompts (A.5: definitions, length strata, and overlaps). Necessity is relative to the finite pool; the score measures behavior, not the policy's internal algorithm.

4 Experiments

4.1 From compositional reachability to policy learning

We evaluate whether a fixed rollout pool contains solutions absent from individual episodes, whether training on these compositions improves held-out behavior, and how composition and training budgets contribute to the results. Blocksworld evaluates symbolic planning, Countdown evaluates arithmetic goal reaching, and ALFWorld evaluates interactive task completion under aliased observation keys (Table 3, A.1). Within each environment, Won-only, Hindsight, and GraphStitch use the same collection pool. Blocksworld and ALFWorld use nested datasets; Countdown uses independently extracted datasets (§4.3).

We report greedy success and pass@8 (T = 1.0, top-p 0.95). Statistical comparisons use task-level paired exact McNemar tests, separately by seed and environment. Appendix A.6 defines the comparison families and corrections; correction-sensitive results are identified below.

Implementation. Supervised conditions share the backbone, LoRA configuration (Hu et al., 2022) (r64/α128), AdamW optimizer, cosine schedule, and bf16 precision within each environment (A.1).

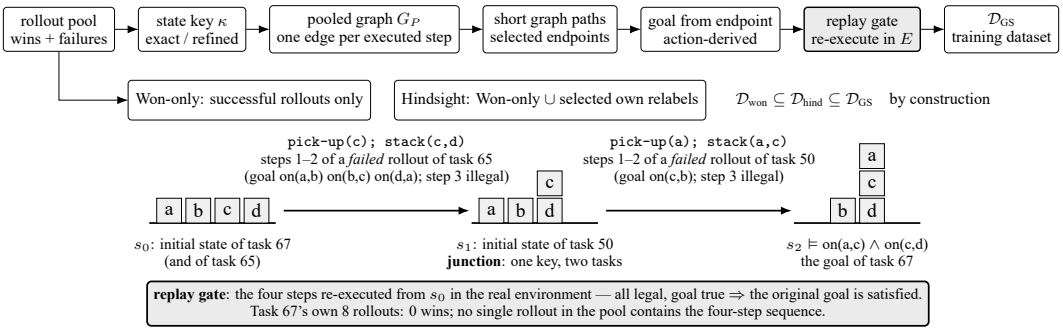


Figure 3: Every step was observed; the complete solution was not. *Top:* the union-based constructor and the two single-episode baselines it contains. *Bottom:* a replay-checked composition from the Blocksworld pool: no rollout of task 67 succeeded, no single rollout contains the four-step sequence, two failed rollouts of *other* tasks meet at a shared state key, and the replay from s_0 verifies the walk (98 such tasks, §4.2).

Blocksworld and ALFWorld use three epochs over each condition's dataset; Countdown uses a shared token budget. Dataset sizes, collection costs, and replay costs are reported separately (A.2–A.4), with budget controls in §4.5.

4.2 Blocksworld: solutions absent from individual episodes

Setup. The state key contains the complete symbolic state. BFS retains one shortest path per eligible endpoint, with depth cap 16 (A.1). The collection pool B0 contains 301 tasks with 3–4 blocks and eight rollouts per task: 2,408 episodes from Qwen3.5-4B-Base using a shared few-shot prompt. Evaluation uses 142 unseen, oracle-solvable 5–6-block instances; no held-out object set occurs in training. We report both training seeds, 20260871 and 20260873.

Solutions supported by the pool. All 13,845 graph paths execute in replay. Removing 6,279 holding-state endpoints and 340 paths through first-hit truncation and deduplication leaves **7,226** paths (A.2). Of these, **83.9% require multiple source episodes**, and **98 of the 301 collection tasks have a solution only after composition**: no individual pool episode provides an original-goal solution from the corresponding initial state. GraphStitch adds 6,717 demonstrations to Hindsight, yielding 7,274 trajectories, or 13.1 times as many. Figure 3 illustrates one solution assembled from two failed rollouts.

Learning from these solutions (Table 1; Figure 2). We evaluate the policies on larger instances without searching the experience graph. GraphStitch improves greedy success over Hindsight by **26.8/29.6pp** across the two seeds, with discordant task counts of 44-vs-6 and 58-vs-16 (p = 3.24e-8/9.67e-7). The gains over Won-only are 52.1/30.3pp. The ordering base < Won-only < Hindsight < GraphStitch holds for both metrics and seeds, although Won-only and Hindsight differ by only one task in seed 2's greedy evaluation. Section 4.5 examines the effect of training volume.

Table 1: Blocksworld. 142 held-out 5–6-block tasks; cells: seed 20260871 / 20260873. Middle rows: the junction ablation — Zero-junction = Hindsight + single-episode-derivable walks, Strict-stitch = Hindsight + junction-necessary walks (§4.2).

condition	trajectories	greedy	pass@8
base	—	1.4%	4.9%
Won-only (RFT)	72	19.0 / 34.5	33.8 / 48.6
Hindsight	557	44.4 / 35.2	59.2 / 54.2
Zero-junction	1,208	39.4 / 46.5	54.2 / 64.1
Strict-stitch	6,623	59.9 / 71.8	75.4 / 87.3
GraphStitch (ours)	**7,274**	**71.1 / 64.8**	**84.5 / 83.1**

Contribution of cross-episode composition. We partition the 6,717 added trajectories into 651 derivable within single episodes and 6,066 requiring multiple episodes. Zero-junction adds the first subset to Hindsight; Strict-stitch adds the second. Both retain all Hindsight data and use the same training and evaluation settings. Mean greedy success is 43.0% for Zero-junction, 65.8% for Strict-stitch, and 68.0% for GraphStitch. Strict-stitch recovers **92.5% of the mean GraphStitch–Hindsight improvement**, with substantial seed variation (58% and 124%). These results support a contribution from composition; the subsets also differ in trajectory count, length, and goal distribution.

GraphStitch exceeds Zero-junction in both seeds (p = 5.7e-10/1.6e-3). Zero-junction is 4.9pp below Hindsight in seed 1 and 11.3pp above it in seed 2 (p = 0.31/9.0e-3); neither comparison survives Bonferroni. Strict-stitch exceeds Hindsight in both seeds (p = 7.2e-3/2.4e-11), but only the second survives correction. It is 2.1pp below GraphStitch on average, with neither per-seed difference significant after correction. A token-matched GraphStitch subset also exceeds Zero-junction by 18pp mean greedy despite containing fewer total dataset tokens (A.2).

4.3 Countdown: arithmetic goal reaching

Setup. Countdown experiments use Qwen3.5-2B-Base, eight rollouts per task, three training seeds, and a shared token budget. State keys are sorted remaining-number tuples; targets are separate goals. GraphStitch is extracted independently, rather than unioned with Hindsight, with at most ten original-goal and ten relabeled paths per task (A.3). The original 600-task evaluation set contains 83 inputs overlapping with the union of GraphStitch and Hindsight relabeled training inputs. We report the original evaluation and a post-hoc analysis excluding those inputs.

Table 2: Countdown (2B backbone). Original 600-task evaluation; 83 inputs overlap with G/HA relabeled training inputs (517-input results: Table 6). Three-seed means, shared token budget. † includes format collapse (A.3). H-only = relabels without wins; TG stitch = recombination toward original targets only (§4.3, A.3).

condition	pass@8	greedy
base	6.33	1.33
Won-only (RFT)	19.17†	12.61†
Hindsight (= HA)	38.78	7.22
H-only (diag.)	40.78	8.11
TG stitch (diag.)	47.33	19.00
GraphStitch (= G)	**48.89**	13.72

Results (Table 2). All reported means include seeds 20260814, 20260815, and 20260816. GraphStitch reaches 48.9% pass@8 and 13.7% greedy, compared with Hindsight's 38.8/7.2% and Won-only's 19.2/12.6%. GraphStitch exceeds Hindsight in every seed at both metrics, with mean gains of **10.1pp pass@8 and 6.5pp greedy**. All six comparisons also remain significant after Bonferroni correction on the 517 non-overlapping inputs; A.3 gives all per-seed rates and paired tests.

Won-only seed 20260814 has format collapse and zero success. Excluding this seed from both methods, GraphStitch retains its pass@8 advantage (48.8% versus 28.8%), while Won-only leads under greedy decoding (18.9% versus 12.8%). Appendix A.3 reports the per-seed sensitivity results, original-goal-only stitching, and the RL diagnostic.

4.4 ALFWorld: composition under aliased observations

Setup. ALFWorld (Shridhar et al., 2021) observations alias distinct states (A.4). We refine observation keys with admissible-command sets and replay candidate paths. The pool contains 19,200 episodes: eight rollouts on each of 2,400 tasks across 53 floorplans, using Qwen3.5-4B-Base. Won-only, Hindsight, and GraphStitch contain 1,960, 16,548, and 33,476 trajectories. Evaluation uses 140 valid_seen and 134 valid_unseen tasks, with no game-file overlap with training, and one training seed.

Held-out results (Table 7, A.4). GraphStitch and Hindsight exceed Won-only on seen greedy success (p = 2.8e-4 and 4.3e-3, respectively; only the first survives correction). We do not detect a GraphStitch–Hindsight difference in any of the four seed-fixed evaluation comparisons. On unseen greedy, GraphStitch and Won-only have equal success counts, with 9-vs-9 discordant tasks. Pass@8 exceeds 92% for every trained condition. Appendix A.6 describes the sample-size and power limits.

Collection-task results. We also evaluate all four policies greedily on the original goals of the 2,400 collection tasks. Among 1,357 tasks with neither a successful collected rollout nor a supplied original-goal solution, Hindsight reaches 70.4% versus Won-only's 60.5% (p = 3.8e-13). GraphStitch reaches 70.9%, with no detected improvement over Hindsight (p = 0.69). On the separate 46 tasks for which construction supplies an original-goal solution, GraphStitch solves 46/46 (A.4).

Compositions and replay validity (Figure 4). Observation-only keys, evaluated on a separate 4,800-rollout pool, leave 1 of 372 non-duplicate cross-episode candidates executable (A.4). With refined keys, **14.94%** of uniformly sampled candidates execute (2,868/19,200); every rejection occurs after the first action. Survival decreases from 24.9% at length three to 5.7% at length ten. Adaptive

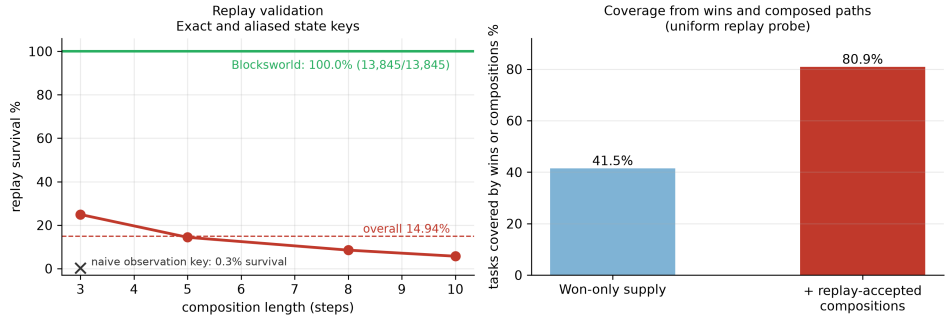


Figure 4: Replay validation on ALFWorld. *Left:* candidate survival decreases with path length, from 24.9% at 3 steps to 5.7% at 10. Reference lines show Blocksworld (100.0%) and the naive observation key (0.3%: 1 of 372 non-duplicate candidates on a separate 4,800-rollout pool, A.4). *Right:* adding verified compositions to successful rollouts increases task coverage at the uniform 8-replay probe. These bars compare wins with wins plus compositions; single-episode hindsight relabeling already covers 2,399/2,400 tasks (§4.4; A.4).

candidate collection supplies an executable cross-episode path for 1,400 of the 1,403 tasks with no successful rollout, including the 46 original-goal solutions above. Hindsight already provides relabeled demonstrations for 2,399/2,400 tasks. Executable paths and endpoint coverage therefore do not by themselves predict the additional learning benefit. Replay checks action legality; A.4 describes the synthetic-goal label errors retained in both datasets.

4.5 Composition and Training Budgets

We vary collection and training budgets to examine alternative explanations for the gains. Blocksworld uses nested collection subsets of 8, 4, 2, and 1 rollouts per task, rebuilding each condition at every tier. Token-matched GraphStitch is tested at the full and one-rollout tiers; the latter also includes a third seed (A.2).

Total dataset-token budget. We select GraphStitch trajectories in deterministic hash order without exceeding Hindsight's total dataset-token count (prompt plus assistant tokens; A.2). At the full collection budget, this subset exceeds Hindsight by **24.6/18.3pp greedy**; both paired comparisons survive correction. Its descriptive pass@8 is 86.6/87.3%, compared with full GraphStitch's 84.5/83.1%. Necessary compositions account for 84.1% of the selected tokens. This control does not match loss-bearing assistant tokens: the matched subset contains 16,370 versus Hindsight's 8,379 (1.95×). At one rollout per task, matched GraphStitch and Hindsight both reach 20.9% mean greedy success. Appendix A.2 distinguishes dataset-token counts from training exposure.

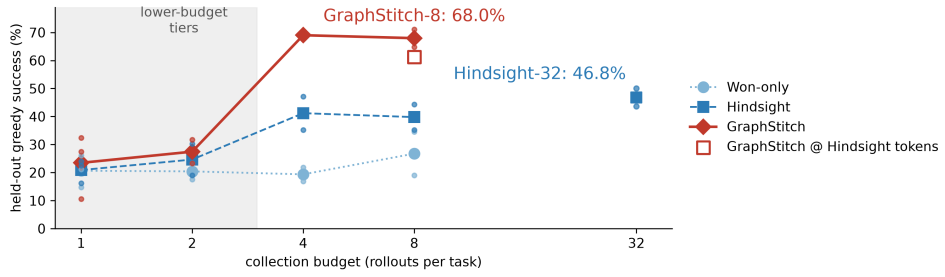


Figure 5: Effect of collection budget on Blocksworld. Held-out greedy success (142 tasks) against collection budget (lines: seed means; dots: seeds). The isolated Hindsight-32 point shows the additional-collection result; no GraphStitch-32 condition was evaluated. GraphStitch-8 achieves 68.0% versus Hindsight-32's 46.8%. Open square: GraphStitch matched to Hindsight's total dataset-token budget. The horizontal axis excludes construction replay (§4.5).

Collection budget. The GraphStitch–Hindsight mean greedy gap is 28.2pp at eight rollouts per task and 27.8pp at four (Figure 5). GraphStitch itself reaches 68.0% and 69.0%, respectively. At two rollouts per task, GraphStitch still contains 7.2 times as many trajectories as Hindsight, but the gap is small and seed-dependent. More constructed trajectories do not yield a comparable gain at this

tier. Per-tier graph connectivity was not measured, and these results do not distinguish coverage, data quality, or goal-distribution explanations.

Composition versus additional collection. GraphStitch using eight collected rollouts per task (2,408 total) achieves 68.0% mean greedy success on the 142 held-out Blocksworld tasks, compared with 46.8% for Hindsight using 32 rollouts per task (9,632 total). Thus, GraphStitch achieves higher held-out success with one-quarter as many collected rollouts. Both training seeds favor GraphStitch ($p = 4.0e\text{-}8$ and $6.5e\text{-}3$; only the first survives Bonferroni correction). This comparison concerns rollout collection rather than total simulator use: including construction replay, GraphStitch and Hindsight require 17,541 and 14,818 invocations, respectively (A.2).

Replay filtering. Every generated Blocksworld path passes replay, so removing that check retains the same dataset. For ALFWorld, the no-gate variant adds all 20,711 selected candidate paths to Hindsight; 17,000 fail replay and account for 84% of the added tokens. On the 1,357 collection tasks defined in §4.4, no-gate reaches 71.4%, versus GraphStitch's 70.9% and Hindsight's 70.4% ($p = 0.69/0.41$). No held-out difference is detected against either condition (largest absolute gap 3.7pp; all $p \geq 0.30$). The variants also differ in candidate selection and training observations, so it evaluates the combined data-construction variant (A.4).

4.6 Behavioral probe: realizing compositional reachability

The probe evaluates goals supported by the experience graph (§3.7). On 600 increment pairs, Hindsight and GraphStitch achieve 50.3% and 94.2% success. These goals are defined relative to a separate RL rollout pool, whereas both supervised policies train on B0 (A.5).

A stricter **post-hoc** analysis requires that no source episode provides a goal-achieving continuation and excludes inputs identical to GraphStitch training prompts. It retains 318 task-indexed pairs but only 122 distinct inputs. Averaging recorded outcomes within each input and then across inputs gives **32.8% and 89.1% input-macro success** for Hindsight and GraphStitch (A.5). These diagnostic results measure goal reaching relative to the source pool. Appendix A.5 reports task-indexed counts and training-goal overlap limitations.

The cold-start GRPO diagnostic reaches 8.5% on the increment probe and 1.6% on the distinct-input strict subset. It samples with action constraints and top-p but scores updates with unconstrained model probabilities. These runs therefore characterize that implementation rather than establish an algorithm-level limitation of GRPO. Training behavior and budgets are reported in A.5.

5 Discussion

What does the collected experience make possible? GraphStitch turns selected paths from the pool's compositional support into demonstrations. Learning from them improves held-out goal reaching in Blocksworld and Countdown. Partition and budget controls support the contribution of composition; goal diversity and single-episode additions may contribute, and ablation effects vary across seeds (§4.2).

From reachable solutions to learned behavior. Graph reachability does not guarantee policy success. GraphStitch evaluates policy success after SFT on constructed demonstrations. ALFWorld marks a boundary: most candidate paths fail replay, and surviving data yield no detectable gain over hindsight. The gate-removal variant also yields no detected benefit or harm (§4.4–4.5). Here, "undemonstrated" refers to the original collection pool, before construction and replay.

Scope and limitations. Experiments cover deterministic environments and 2B/4B backbones. Blocksworld lacks matched in-distribution evaluation; Countdown uses independently extracted datasets and has training-input overlap in the original evaluation set. ALFWorld has one seed, limited evaluation sets (A.6), and synthetic-goal label limitations (A.4). Additional collection and no-gate each use one budget. The Blocksworld RL diagnostic uses one cold-start recipe; broader comparisons of RL algorithms, budgets, and initialization remain future work, as does iterative composition and learning.

Reproducibility Statement

Appendix A.1 specifies the environments, data selection, training, and decoding settings. A.2–A.5 give additional results, budget accounting, and probe definitions; A.6 describes the statistical protocol. Experiment-specific limitations are stated alongside the corresponding results. Code will be made publicly available.

The Use of Large Language Models

LLMs are the policies studied here and generate the interaction trajectories used to construct training data (§3–§4). Generative AI assistants also supported literature search, research framing and experimental review, code development and analysis, interpretation of results, figure preparation, and manuscript drafting and revision, including the derivation in Appendix A.6. The author takes responsibility for the paper's content.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. *arXiv preprint arXiv:1707.01495*, 2017.

Seungho Baek, Taegeon Park, Jongchan Park, Seungjun Oh, and Yusung Kim. Graph-Assisted Stitching for Offline Hierarchical Reinforcement Learning. In *International Conference on Machine Learning*, 2025. arXiv:2506.07744.

Michał Bortkiewicz, Władysław Pałucki, Mateusz Ostaszewski, and Benjamin Eysenbach. Is Temporal Difference Learning the Gold Standard for Stitching in RL? *arXiv preprint arXiv:2510.21995*, 2025.

David Brandfonbrener, Alberto Bietti, Jacob Buckman, Romain Laroche, and Joan Bruna. When does return-conditioned supervised learning work for offline reinforcement learning? *arXiv preprint arXiv:2206.01079*, 2022.

Ian Char, Viraj Mehta, Adam Villaflor, John M. Dolan, and Jeff Schneider. BATS: Best Action Trajectory Stitching. *arXiv preprint arXiv:2204.12026*, 2022.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling. *arXiv preprint arXiv:2106.01345*, 2021.

Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Sara Cao, Yuanhao Xiong, Haibo Tong, Huaxiu Yao, Hengduo Li, Jiacheng Zhu, Xian Li, Dawn Song, Bo Li, Jason Weston, and Dat Huynh. Scaling Agent Learning via Experience Synthesis. *arXiv preprint arXiv:2511.03773*, 2025.

Peter Dayan and Geoffrey E. Hinton. Using Expectation-Maximization for Reinforcement Learning. *Neural Computation*, 9(2):271–278, 1997.

Liang Ding. AgentHER: Hindsight Experience Replay for LLM Agent Trajectory Relabeling. *arXiv preprint arXiv:2603.21357*, 2026.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-Explore: a New Approach for Hard-Exploration Problems. *arXiv preprint arXiv:1901.10995*, 2019. Journal version: First return, then explore. Nature 590:580–586, 2021.

Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Search on the Replay Buffer: Bridging Planning and Reinforcement Learning. *arXiv preprint arXiv:1906.05253*, 2019.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-Group Policy Optimization for LLM Agent Training. *arXiv preprint arXiv:2505.10978*, 2025.

Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to Reach Goals via Iterated Supervised Learning. In *International Conference on Learning Representations*, 2021. arXiv:1912.06088.

Raj Ghugare, Matthieu Geist, Glen Berseth, and Benjamin Eysenbach. Closing the Gap between TD Learning and Supervised Learning – A Generalisation Point of View. In *International Conference on Learning Representations*, 2024. arXiv:2401.11237.

Charles A. Hepburn and Giovanni Montana. Model-based Trajectory Stitching for Improved Offline Reinforcement Learning. *arXiv preprint arXiv:2211.11603*, 2022.

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. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=nZeVKeeFYf9`.

Yifu Huo, Shunjie Xing, Chenglong Wang, Peinan Feng, Qiaozhi He, Yan Ding, Anxiang Ma, Yuxin Gao, Tongran Liu, Tong Xiao, and Jingbo Zhu. Learning from Environmental Feedback: Credit Assignment across Multiple Timescales for Agentic Reinforcement Learning. *arXiv preprint arXiv:2608.08255*, 2026.

Li-Cheng Lan, Andrew Bai, Minhao Cheng, Cho-Jui Hsieh, and Tianyi Zhou. Exploring Expert Failures Improves LLM Agent Tuning. *arXiv preprint arXiv:2504.13145*, 2025.

Daniel Lawson, Adriana Hugessen, Charlotte Cloutier, Glen Berseth, and Khimya Khetarpal. Self-Predictive Representations for Combinatorial Generalization in Behavioral Cloning. *arXiv preprint arXiv:2506.10137*, 2025.

Xuanyu Lei, Yiqi Zhu, Chenliang Li, Kaiming Liu, Peng Li, Ming Yan, Jieping Ye, Ya-Qin Zhang, and Yang Liu. State2State: Environment-Derived Mid-Training for LLM Agents. *arXiv preprint arXiv:2608.04934*, 2026.

Guanghe Li, Yixiang Shan, Zhengbang Zhu, Ting Long, and Weinan Zhang. DiffStitch: Boosting Offline Reinforcement Learning with Diffusion-based Trajectory Stitching. *arXiv preprint arXiv:2402.02439*, 2024.

Jiazheng Li, Yawei Wang, Qiaojing Yan, Yijun Tian, Zhichao Xu, Huan Song, Panpan Xu, and Lin Lee Cheong. SALT: Step-level Advantage Assignment for Long-horizon Agents via Trajectory Graph. In *Findings of the Association for Computational Linguistics: EACL 2026*, 2026a. arXiv:2510.20022.

Zichao Li, Gang Wu, Zichao Wang, Ruiyi Zhang, Wanrong Zhu, Ryan A. Rossi, Vlad I. Morariu, and Jihyung Kil. Spinning Straw into Gold: Relabeling LLM Agent Trajectories in Hindsight for Successful Demonstrations. *arXiv preprint arXiv:2607.04235*, 2026b.

Jiaye Lin, Yifu Guo, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, Mingguang Chen, Hongzhang Liu, Ronghao Chen, Yangfan He, Daxin Jiang, Binxing Jiao, Chen Hu, and Huacan Wang. SE-Agent: Self-Evolution Trajectory Optimization in Multi-Step Reasoning with LLM-Based Agents. *arXiv preprint arXiv:2508.02085*, 2025.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-Like Training: A Critical Perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Cong Lu, Shengran Hu, and Jeff Clune. Intelligent Go-Explore: Standing on the Shoulders of Giant Foundation Models. *arXiv preprint arXiv:2405.15143*, 2024.

Simon Park, Simran Kaur, and Sanjeev Arora. How Does RL Post-training Induce Skill Composition? A Case Study on Countdown. *arXiv preprint arXiv:2512.01775*, 2025.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. Agent Q: Advanced Reasoning and Learning for Autonomous AI Agents. *arXiv preprint arXiv:2408.07199*, 2024.

Qwen Team. Qwen3.5: Towards Native Multimodal Agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*, 2024.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.03768.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models. *Transactions on Machine Learning Research*, 2024. arXiv:2312.06585.

Charlie Snell, Ilya Kostrikov, Yi Su, Mengjiao Yang, and Sergey Levine. Offline RL for Natural Language Generation with Implicit Language Q Learning. *arXiv preprint arXiv:2206.11871*, 2022.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and Error: Exploration-Based Trajectory Optimization for LLM Agents. *arXiv preprint arXiv:2403.02502*, 2024.

Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. STeCa: Step-level Trajectory Calibration for LLM Agent Learning. *arXiv preprint arXiv:2502.14276*, 2025a.

Shuze Wang, Yunpeng Mei, Hongjie Cao, Yetian Yuan, Gang Wang, Jian Sun, and Jie Chen. Robust Offline Imitation Learning Through State-level Trajectory Stitching. *arXiv preprint arXiv:2503.22524*, 2025b.

Yunan Wang, Minghui Song, Zihan Zhang, Shaohan Huang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. Group-Graph Policy Optimization for Long-Horizon Agentic Reinforcement Learning. *arXiv preprint arXiv:2606.22995*, 2026a.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN: Understanding Self-Evolution in LLM Agents via Multi-Turn Reinforcement Learning. *arXiv preprint arXiv:2504.20073*, 2025c.

Ziting Wang, Yin Li, Zuhao Yang, Xiuchang Li, Jiale Bai, and Gao Cong. Demonstrating TOFFEE: A Learned System for Synthesizing Data Agent Trajectories at Scale. *arXiv preprint arXiv:2607.06233*, 2026b.

Guowei Xu, Zhenting Qi, Huangyuan Su, Weirui Ye, Himabindu Lakkaraju, Sham M. Kakade, and Yilun Du. Self-Improving Language Models with Bidirectional Evolutionary Search. *arXiv preprint arXiv:2605.28814*, 2026.

Taku Yamagata, Ahmed Khalil, and Raul Santos-Rodriguez. Q-learning Decision Transformer: Leveraging Dynamic Programming for Conditional Sequence Modelling in Offline RL. *arXiv preprint arXiv:2209.03993*, 2022.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. DAPO: An Open-Source LLM Reinforcement Learning System at Scale. *arXiv preprint arXiv:2503.14476*, 2025.

Xiao Yu, Baolin Peng, Vineeth Vajipey, Hao Cheng, Michel Galley, Jianfeng Gao, and Zhou Yu. ExACT: Teaching AI Agents to Explore with Reflective-MCTS and Exploratory Learning. *arXiv preprint arXiv:2410.02052*, 2024.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling Relationship on Learning Mathematical Reasoning with Large Language Models. *arXiv preprint arXiv:2308.01825*, 2023.

Yuliang Zhan, Xinyu Tang, Jian Li, Dandan Zheng, Weilong Chai, Jingdong Chen, Jun Zhou, Ge Wu, Wenyue Tang, and Hao Sun. GraphPO: Graph-based Policy Optimization for Reasoning Models. *arXiv preprint arXiv:2606.18954*, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language Agent Tree Search Unifies Reasoning Acting and Planning in Language Models. *arXiv preprint arXiv:2310.04406*, 2023.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. ArCHer: Training Language Model Agents via Hierarchical Multi-Turn RL. *arXiv preprint arXiv:2402.19446*, 2024.

A Appendix

A.1 Experimental configuration

Table 3 summarizes the environments and evaluation splits; Table 4 gives the training and decoding settings. Won-only, Hindsight, and GraphStitch share one collection pool per environment. The two Blocksworld training seeds therefore measure variation conditional on the same pool, rather than variation across independent collections.

State keys and path selection. Blocksworld uses the complete symbolic state and rejects conflicting successors for a (key, action) pair. Breadth-first search retains one shortest path per reachable endpoint, up to depth 16. Relabeled goals describe all on/on-table placements at hand-empty endpoints different from the initial state; original-goal paths end at the first goal hit. Replay checks action legality and agreement with the predicted endpoint key. Training uses 301 tasks with 3–4 blocks and oracle depths 2, 4, or 6; the 142 held-out tasks use 5–6 blocks, so no held-out object set occurs in training. Countdown keys sorted remaining-number tuples, with targets stored separately, and retains at most ten original-goal and ten relabeled paths per task (A.3). ALFWorld keys the observation and admissible-command set, builds graphs within floorplans, and replays selected candidates (A.4).

Hindsight selection. Blocksworld retains a failed rollout's maximal nonempty legal prefix when its endpoint meets the placement criteria above. ALFWorld retains the longest prefix ending at an action-derived achievement. Countdown retains legal failed trajectories of the prescribed three-action length that reach a goal-sized state. Each condition adds successful demonstrations. These rules select eligible prefixes; they do not enumerate all intermediate prefixes.

Training text. Blocksworld uses complete plans under the same few-shot prompt for collection, SFT, and evaluation. Appendix A.7 gives a complete rendered prompt, including both examples and the final task. Its system message requests numbered actions, whereas the examples and final user message request action strings; the decoder enforces action strings. This mixed-format context is shared across conditions. ALFWorld packs each trajectory as one conversation, includes admissible commands at every user turn, and applies loss to assistant actions only. Training and evaluation share the renderer. Long conversations are split with a 6,000-token target while preserving the history needed for each supervised turn; this is not a hard cap (recorded maxima are 10,184/12,580/12,580 tokens for Won-only/Hindsight/GraphStitch).

Blocksworld action trie. For each problem, the decoder builds a token trie from the sorted, statically grounded action table over its objects, plus STOP. Separate tokenizations cover the first action and space-prefixed subsequent actions. After an action ends, the trie resets to its continuation root; STOP permits termination. The same table is used throughout a generated plan. It includes actions whose preconditions are false and is not updated from simulated states. Thus the trie constrains syntax and object names, not action legality. The complete plan is generated without intermediate

Table 3: The three environments. Won-only, Hindsight, and GraphStitch share a collection pool within each environment (§3.5). The gate column reports Blocksworld executability and ALFWorld's uniformly sampled relabel probe.

environment	state key	pool (tasks × rollouts)	evaluation set	seeds	gate survival
Blocksworld	full symbolic state (exact)	$301 \times 8 = 2{,}408$	142 tasks, 5–6 blocks	2	**100.0%**
Countdown	remaining numbers (exact)	$1{,}200 \times 8 = 9{,}600$	600 tasks	3	exact key
ALFWorld (alias key)	observation + admissible set	$2{,}400 \times 8 = 19{,}200$	140 seen / 134 unseen	1	**14.94%**

Supervised conditions (Won-only, Hindsight, GraphStitch, and every ablation or control trained by SFT)	
Backbone	Qwen3.5-4B-Base (Blocksworld, ALFWorld); Qwen3.5-2B-Base (Countdown) (Qwen Team, 2026)
Adapter	LoRA rank 64, α 128, dropout 0.05, no bias, on q/k/v/o_proj, in_proj_{a, b, qkv, z}, out_proj, gate/up/down_proj
Optimizer	AdamW, lr 10^{-4}, weight decay 0.01, $\beta = (0.9, 0.999)$; cosine schedule with warmup $\max(5\%, 3)$ steps; bf16; gradient clip 1.0; gradient checkpointing
Batch, loss	16 sequences per update (micro-batch × accumulation = 16); loss on assistant tokens only
Budget	three nominal epochs over each condition's own dataset (Blocksworld, ALFWorld); Countdown uses 5× the TG-stitch dataset's tokens (effective epochs Won-only 16.7 / Hindsight HA 1.06 / GraphStitch 0.48); whole-batch truncation slightly undershoots token ceilings
Seeds	Blocksworld {20260871, 20260873} (+20260877 at the 1/8 tier); ALFWorld 20260871; Countdown {20260814, 20260815, 20260816}
On-policy control (GRPO, §4.6; implementation limits in A.5)	
Estimator	Reward minus group mean, without reward-standard-deviation normalization; summed token surrogate divided by batch size times 160 (192 on Countdown), following Dr. GRPO (Liu et al., 2025); group size 8, 4 groups per step (2 on Countdown), two PPO epochs, clip 0.20 / 0.28, no KL term
Sampling	T = 1.0, top-p 0.95; DAPO-style dynamic sampling (Yu et al., 2025) (all-solved and all-failed groups discarded; 2.5× oversampling, up to 6 resample rounds)
Optimizer	AdamW, lr 10^{-4}, weight decay 0.01, gradient clip 1.0, bf16; the same LoRA modules with dropout 0; cold start from the base except for one Countdown run initialized from a stitching-SFT checkpoint (A.3)
Budget	stated ceilings 9,634 (two seeds) and 2,408; actual episodes sampled, kept or discarded: 9,632 and 2,408
Evaluation (shared decoding within each environment; success evaluated by the environment)	
Decoding	greedy (T = 0), one rollout; pass@8: eight samples at T = 1.0 / top-p 0.95
Episode caps	Blocksworld: one plan of ≤160 new tokens; Countdown: ≤6 turns × 32 tokens; ALFWorld: ≤30 turns × 32 tokens, served by SGLang 0.5.17 from the merged weights

Table 4: Training and evaluation configuration. SFT conditions share the recipe within each environment; data size and budget controls vary as specified. GRPO uses the separate configuration shown here (§3.6).

environment feedback, then executed; an invalid action causes failure. Collection, SFT evaluation, and the behavioral probe use this same construction.

A.2 Blocksworld training data and budget controls

Composition and single-episode controls. The graph contains 1,402 states and 1,425 edges, with no conflicting successors. All 13,845 generated walks execute; removing 6,279 holding endpoints and 340 paths after first-hit truncation/deduplication leaves 7,226 walks. Of these, 6,066 require multiple source episodes, 1,085 are prefixes of a selected episode, and 75 are non-prefix, order-preserving subsequences of one episode's (state, action) pairs, obtained by trimming its ends and removing intervening return-to-state loops. Same-task sources are preferred when available. The multi-episode walks require 2–9 sources (mean approximately four). GraphStitch adds 6,717 walks after excluding 509 already present in Hindsight under the (task, goal, actions) key. Zero-junction adds the 651 new single-episode walks, while Strict-stitch adds all 6,066 necessary compositions: their union is GraphStitch and their intersection is Hindsight. Original-goal task coverage is 71/84/182 for

Table 5: Training datasets. Counts are complete trajectories before per-turn rendering or conversation packing. Countdown uses independently extracted datasets (A.3); Blocksworld and ALFWorld use the nested construction in §3.5.

	Countdown	Blocksworld	ALFWorld
Collection episodes	9,600	2,408	19,200
Won-only	167	72	1,960
Hindsight	2,626	557	16,548
GraphStitch	5,876	7,274	33,476

Hindsight/Zero-junction/GraphStitch. For 98 of the 301 tasks, no source episode, after any visit to the task's initial state, reaches a state satisfying the original goal; GraphStitch supplies a composed solution.

An exhaustive prefix scan yields 846 eligible demonstrations, 289 absent from Hindsight. Every prefix trajectory, task–goal pair, or initial-state–goal pair added by GraphStitch is also present in Zero-junction; both omit the same 89 prefix trajectories. This also holds under original-goal priority and first-hit truncation. Thus Zero-junction controls the single-episode prefix additions made by GraphStitch. An exhaustive-prefix baseline was not trained.

Training budgets. Token matching selects rows in deterministic hash order using SHA-256 of the UTF-8 string '20260871:sample_id'. A row is selected if its prompt-plus-assistant token count fits the remaining Hindsight budget; otherwise it is skipped and selection continues. Selected rows are then sorted by sample ID. Hash ordering avoids selecting only the inherited Hindsight rows, which occur first in the serialized GraphStitch data. The matched subset has 537 trajectories over 172 tasks; necessary compositions account for 83.6% of trajectories and 84.1% of tokens. Total-token matching does not match assistant tokens contributing to the loss:

Condition	Trajectories	Total tokens	Assistant tokens	Mean actions
Hindsight	557	621,890	8,379	3.00
Zero-junction	1,208	1.36M	17,115	2.76
Token-matched GraphStitch	537	621,672	16,370	6.50

Counts are for one pass through the data. The matched condition achieves 69.0/53.5% greedy and 86.6/87.3% pass@8 across the two seeds. Zero-junction has more assistant tokens than matched GraphStitch but lower success; trajectory length and goal distributions remain different. The partition and token controls do not independently identify each of these contributions.

Collection controls. Scarcity tiers use nested subsets of B0 at 8, 4, 2, and 1 rollouts per task, rebuilding every condition at each tier. The GraphStitch–Hindsight mean greedy gap is 28.2/27.8 pp at 8/4 rollouts, then becomes small and seed-dependent at 2/1 rollouts (Figure 5). Per-tier graph connectivity was not measured, so the curve does not establish a connectivity threshold. The additional-collection control merges B0 with three independent eight-rollout collections under the same prompt (collection seeds 2026081404/05/06). Hindsight-32 has 1,300 demonstrations and reaches 43.7/50.0% greedy and 54.9/66.9% pass@8. Its two comparisons with GraphStitch have greedy p = 4.0e-8/6.5e-3; only the first survives the reported correction. Collection and construction costs are:

Condition	Collection episodes	Construction replays	Total invocations
Hindsight-8	2,408	1,288	3,696
Hindsight-32	9,632	5,186	14,818
GraphStitch-8	2,408	13,845 + 1,288	17,541

Graph replays execute 89,045 symbolic actions. These counts exclude initial-state setup, evaluation, and diagnostics; episode lengths, generation costs, and simulator costs differ. Construction costs count all executed replays, including paths excluded from training. Neither Hindsight-32 nor the 9,632-episode RL run matches GraphStitch's measured total simulator use.

A.3 Countdown configuration and results

The Countdown GraphStitch condition G contains 5,876 independently extracted trajectories. Hindsight HA combines 167 successful and 2,459 failed-trajectory demonstrations, rendered as 7,878 per-turn examples. G is not the union of HA with additional paths. The shared training budget is 4,284,120 tokens, equivalent to 16.7/1.06/0.48 epochs of Won-only/HA/G; original-goal stitching uses five epochs. We report pass@8 and greedy success. The environment, backbone, and construction procedure differ from Blocksworld; the comparison across environments does not isolate the effect of backbone size.

Training-input overlap. Of the original 600 evaluation tasks, 75 share a (start, goal) pair with a G relabel and 26 with HA. Removing their union leaves 517 tasks. G still exceeds HA in all three seeds at both endpoints; pass@8 p = 6.3e-7/1.7e-3/3.2e-6. These overlap-excluded results support the GraphStitch–Hindsight comparison without treating all 600 tasks as unseen inputs. Table 6 reports per-seed results; the exclusion uses G and HA training inputs, not the union of every baseline.

Table 6: Countdown after excluding overlapping inputs. Results on the same 517 tasks for GraphStitch (GS) and Hindsight (H); rates are percentages and Δ is GS minus H in percentage points. All six paired exact McNemar tests pass Bonferroni correction at $0.05/6 \approx 0.00833$. This is a post-hoc sensitivity analysis of the original 600-task set.

Seed	Metric	H	GS	Δ	p
20260814	pass@8	34.43	46.81	+12.38	6.33×10^{-7}
20260815	pass@8	38.88	46.62	+7.74	1.70×10^{-3}
20260816	pass@8	34.62	46.23	+11.61	3.21×10^{-6}
20260814	greedy	4.84	13.93	+9.09	4.32×10^{-10}
20260815	greedy	5.42	9.09	+3.68	7.94×10^{-3}
20260816	greedy	5.42	10.83	+5.42	3.94×10^{-4}

Countdown seed sensitivity. Table 2 and Figure 2 retain all three seeds. Won-only seed 20260814 has zero success and 4,583 parsing errors in 4,800 sampled episodes. Excluding this seed from both conditions leaves seeds 20260815 and 20260816: Won-only solves 161/184 tasks at pass@8 and 107/120 greedily; GraphStitch solves 292/293 and 72/81, respectively, out of 600 per seed. Their two-seed means are 28.75/48.75% pass@8 and 18.92/12.75% greedy (Won-only/GraphStitch). The greedy ordering reverses relative to the all-seed means; this sensitivity analysis does not replace the three-seed comparison.

Original-goal stitching. This condition composes paths toward original targets without relabeling: 337 of its 560 walks (60.2%) are multi-episode compositions. It achieves 47.33% pass@8 and 19.00% greedy, compared with G's 48.89/13.72% and hindsight-only's 40.78/8.11%. The 231 hard tasks have 1–7 legal solution sequences, determined by exhaustive enumeration. Across three seeds, original-goal stitching solves 68 of these tasks, versus none across the eleven GRPO runs (ten cold-start and one initialized from a stitching SFT checkpoint). This is a union of solved tasks across runs, not a 100% success rate on that stratum.

RL diagnostic. The eleven Countdown GRPO runs (ten cold-start and one initialized from a stitching SFT checkpoint) jointly sample 105,600 episodes using a GRPO-based configuration (Shao et al., 2024). Of their 13,200 groups, 96.3% are all-fail, 2.6% all-success, and 1.1% mixed. Composition-only goals receive 3,152 episodes without a success or nonzero advantage. These pooled counts describe one near-floor 2B configuration; they are not the budget or failure rate of a single interaction-matched run.

A.4 ALFWorld construction, results, and limitations

Table 7: ALFWorld (alias-key environment). Task success under the shared rendering, seed 20260871; valid_seen $n = 140$, valid_unseen $n = 134$. Both recycling conditions clear Won-only on seen greedy with task-paired significance (GraphStitch 21-vs-3, $p = 2.8 \times 10^{-4}$; Hindsight 18-vs-4, $p = 4.3 \times 10^{-3}$, correction-sensitive, §4.1); the GraphStitch–Hindsight difference is not significant in any of the four comparisons. No held-out difference is detected between no-gate GraphStitch (§4.5) and either recycling condition (largest absolute gap 3.7pp, $p \geq 0.30$). These single-seed results do not establish equivalence; power assumptions are given in A.6.

	valid_seen		valid_unseen	
condition	greedy	pass@8	greedy	pass@8
base	21.43%	53.57%	14.93%	55.97%
Won-only	72.86%	92.86%	87.31%	96.27%
Hindsight	82.86%	**97.86%**	90.30%	**98.51%**
GraphStitch	**85.71%**	95.71%	87.31%	97.01%
no-gate GraphStitch	82.86%	97.14%	**91.04%**	**98.51%**

Aliasing and replay. Cross-task shared-edge artifacts occur at 72.9% under observation-only keys and 0.5% after refinement with admissible commands; these are edge diagnostics, not per-junction failure probabilities. The observation-only construction, run on a separate 4,800-rollout pool, produces 399 cross-episode candidates; 28 execute in replay, 27 of which are action-identical to a successful rollout of the same task, so only 1 of the 372 non-duplicate candidates survives (371/372 rejected). With refined keys, 2,868/19,200 uniformly sampled candidates execute (14.94%). All rejections are mid-path inadmissibility, with none at the first action. Survival at lengths 3/5/8/10 is 24.9/14.4/8.5/5.7%. A 500/500 prefix-replay check supports deterministic replay in this setup.

Adaptive candidate collection targets up to eight verified walks per task with a 64-attempt cap. It makes 37,564 additional replays and reuses previously verified paths; these counts therefore cannot estimate the uniform probe's acceptance rate. Of 17,156 collected path records, 16,703 are relabeled and 453 carry original goals. Removing 221 Hindsight duplicates and seven internal duplicates leaves 16,928 additions. Relabeled candidates matching a successful same-task rollout's complete executed action sequence are excluded, regardless of goal label. For relabeling, a candidate is classified as a single-episode prefix when its complete action sequence equals the first n executed world actions of any rollout of the same task, for candidate length n. Only legal actions are compared, excluding 'look', 'inventory', 'examine', and 'help'. These candidates are excluded from adaptive relabel-path selection; prefix replay controls are also excluded from reused survivors. Original-goal candidates are handled separately. No novel prefix-class trajectory remains in the final additions. Hindsight covers 2,399/2,400 tasks. Among 1,403 tasks with no successful collection rollout, the constructor supplies executable cross-episode walks for 1,400; for 46 tasks a walk also wins the original goal. These are different notions of coverage.

A post-hoc endpoint scan finds 6,482 Blocksworld and 14,938 ALFWorld task–endpoint pairs reached by replayed paths but absent from the task's own rollouts. These measure endpoint coverage, not necessarily new goal labels or composition necessity: some Blocksworld pairs can be reached within one foreign episode.

Goal-label limitation. Original task goals use the environment's success signal. Synthetic goals use an action-template scanner, while replay checks action legality. The scanner retains objects ever moved to a receptacle even if they are subsequently removed. Tracking these moves identifies 41 of 876 two-object Hindsight trajectories whose final tracked placements do not support their labels (41/16,548 of all Hindsight trajectories). GraphStitch inherits all 41; none of its 327 new two-object compositions has this tracked inconsistency. This check does not recover the complete latent state. No repaired-label retraining was performed. Replay acceptance thus does not certify every synthetic goal label.

Held-out and collection-task results. GraphStitch–Hindsight discordants are 11/7 and 4/8 for seen/unseen greedy, and 2/5 and 0/2 for pass@8; all four comparisons are nonsignificant. The uncorrected 80%-power detectable greedy differences are 8.3/6.6 pp. Pass@8 saturates above 92% for all trained conditions (Figure 6). The collection-task evaluation uses original goals, excluding the 46 zero-success tasks with a supplied original-goal solution. On the remaining 1,357 tasks, base/Won-

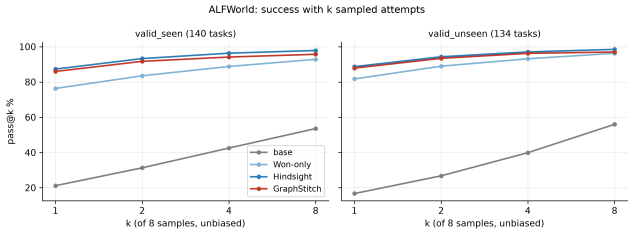


Figure 6: ALFWorld pass@k. Unbiased pass@k from 8 samples per task at T = 1.0, top-p 0.95. Sampled pass@1 differs from greedy success; pass@8 exceeds 92% for every trained condition (§4.4).

only/Hindsight/ GraphStitch achieve 5.8/60.5/70.4/70.9% greedy. GraphStitch–Hindsight has discordants 115/108, p = 0.69; Hindsight–Won-only has 239/105, p = 3.8e-13. The 46 excluded tasks are descriptive: 3/42/44/46 successes, respectively. Full-pool rates are 15.9/75.5/81.0/82.0%.

Gate-removal control. The no-gate condition adds all 20,711 selected compositions to Hindsight, producing 37,259 trajectories. Of these additions, 17,000 fail replay and contain 84.0% of new tokens. It reaches 71.4% on the 1,357-task probe (p = 0.69 versus GraphStitch; p = 0.41 versus Hindsight), and 82.86/97.14% seen and 91.04/98.51% unseen greedy/pass@8. All eight held-out comparisons with GraphStitch and Hindsight have p $\geq$ 0.30. The variants also differ in candidate selection (uniform sampling versus adaptive collection) and training text (donor observations versus replay transcripts). This is consequently a comparison of complete data variants, not an isolated effect of replay verification. No policy benefit or harm is detected at the tested three-epoch budget.

A.5 Behavioral probe and Blocksworld RL diagnostic

Table 8: Cold-start GRPO diagnostic (Blocksworld). Held-out greedy: 142 tasks; GRPO actually samples 9,632 episodes per seed. Sampling/scoring limitations are discussed in A.5. Probe columns use seed 20260871 and the RL run's experience: own (n = 376), increment (n = 600), endpoint-necessary (n = 471), and the post-hoc strict subset (n = 318 task-indexed pairs, 122 distinct inputs) requiring goal-level necessity and excluding identical GraphStitch training inputs (A.5). Strict rates here weight task-indexed records equally; input-macro rates in §4.6 average within and then across input groups. SFT held-out values are two-seed means.

condition	held-out greedy	own	increment	endpoint-necessary	strict (post hoc) task-indexed
base	1.4	31.4	7.5	2.8	0.9
GRPO (9,632 eps)	0.0 / 0.0	55.8	8.5	3.0	1.6
Hindsight	39.8	99.5	50.3	38.6	31.1
GraphStitch	**68.0**	**100**	**94.2**	**92.6**	**89.0**

Pair construction. From every source episode, including discarded RL samples, we build the state-transition graph and search to depth 16 from each task's initial state. Own endpoints occur in that task's rollouts; increment endpoints are graph-reachable but absent from those rollouts. Goals are the endpoint's placement literals; duplicate goals and goals already true initially are removed. We sample at most 600 pairs per set with seed 2026090301 and use the same greedy, trie-constrained decoder for every policy. Success requires legal actions and satisfaction of the goal. For the 9,632-episode RL pool this gives 376 own pairs and 600 sampled increment pairs (from 1,515). SFT trains on B0, whereas this diagnostic is defined relative to the RL pool.

Length and endpoint necessity. Of the increment pairs, 129 can reach their endpoint within one source episode after the initial state, and 471 require composition. Goal predicates may hold at states other than the chosen endpoint. Holding endpoints occur in 237 increment pairs, although GraphStitch relabeling excludes them. Own and increment sets also differ in length. The following cells give own/increment counts by graph-path length, which need not be the shortest goal-achieving length:

Length	Pairs	GRPO wins	GraphStitch wins
1–2	163 / 73	130 / 20	163 / 73
3–4	184 / 142	80 / 24	184 / 142
≥5	29 / 385	0 / 7	29 / 350

Strict post-hoc sensitivity. We check every source episode that visits the initial state for any later state satisfying the goal. Of the 471 endpoint-necessary pairs, 22 have such a goal witness, leaving 449 goal-necessary pairs. Separately, 209/600 increment inputs match GraphStitch training byte-for-byte and 49 match Hindsight. Intersecting goal necessity with no GraphStitch input match leaves 318 pairs: base/GRPO/Hindsight/GraphStitch solve 3/5/99/283. These are task-indexed pairs with only 122 distinct rendered inputs. Table 8 reports task-indexed rates. For input-macro success, we average the binary outcomes of records sharing each input, then average over input groups:

$$\text{Success}_{\text{macro}} = \frac{1}{122} \sum_{x} \frac{1}{|I(x)|} \sum_{i \in I(x)} y_i.$$

Here $I(x)$ indexes records with input x and $y_i \in \{0, 1\}$ denotes success.

Groups use the canonical complete initial state and ordered goal literals; equal keys produce identical prompt messages and static action tables. We retain every recorded outcome within each group, rather than selecting one prediction. This gives base/GRPO/Hindsight/GraphStitch input-macro rates of 0.8/1.6/32.8/89.1%. GraphStitch has 108 uniformly successful groups, 12 uniformly failed groups, and two groups with 2/4 and 1/4 success, respectively: (108 + 0.5 + 0.25)/122 = 89.139%. These repeated inputs were evaluated separately in batched greedy inference and produced different plans despite identical messages and constraints; the cause of this inference variation was not isolated. The other policies' outcomes agree within every repeated-input group. We make no independence-based significance claim for repeated inputs. This post-hoc analysis reuses recorded predictions. Exact-input exclusion does not exclude goals implied by stronger training goals; necessity is relative to the finite source pool.

RL training. The implementation uses reward minus group mean, without division by group reward standard deviation, and divides the summed clipped token surrogate by batch size times a fixed maximum response length (160 for Blocksworld, 192 for Countdown). These are Dr. GRPO-style advantage and loss normalization (Liu et al., 2025), combined with asymmetric clipping and DAPO-style dynamic sampling (Yu et al., 2025) in groups of eight. This combination is not a reproduction of either complete training system. Every sampled episode is counted, including discarded groups. Both seeds at the larger budget sample 9,632 episodes. Seed 20260871 retains 88 episodes and updates on 4/23 steps; seed 20260873 retains 224 (216 enter updates under the four-groups-per-step cap) and updates on 9/24. Both obtain 0/142 held-out greedy success. The second run collapses to whitespace output and is excluded from the behavioral probe. A 2,408-episode run retains 80 episodes, updates on 3/7 steps, and solves 2/142 held-out tasks. Runs exhaust their sampling budgets without stopping early after consecutive steps with no update. Generation applies the action trie and top-p, but PPO ratios use unconstrained model log-probabilities. Their correspondence to the constrained sampling policy, and their role in the collapse, have not been validated. These results diagnose the implemented cold-start configuration; they do not establish a general failure of RL to learn composition. RAGEN reports related training instabilities (Wang et al., 2025c), while other Countdown settings exhibit compositional generalization (Park et al., 2025).

A.6 Statistical protocol and objective interpretation

Paired comparisons. Held-out comparisons use task-level exact McNemar tests separately for each seed and environment. Their significance is conditional on the trained policies; training-seed replication does not account for collection-pool variability. The Blocksworld greedy family contains 18 comparisons: two GraphStitch–Hindsight contrasts at eight rollouts per task, ten partition contrasts, two token-matched contrasts, and four additional-collection contrasts. The Bonferroni threshold is 0.05/18. For these GraphStitch–Hindsight contrasts, discordant counts 44/6 and 58/16 give p = 3.24e-8/9.67e-7. Correction-sensitive comparisons are flagged in §4; pass@8 results are descriptive. Nonsignificance is not interpreted as equivalence.

Countdown original-goal stitching has two comparison targets, Won-only and hindsight-only, with Bonferroni threshold 0.05/2 = 0.025. For GraphStitch versus Hindsight (G versus HA), three seeds and two metrics define six paired tests on the original 600-task set, corrected at 0.05/6. The 517-task overlap-excluded analysis is a separate, explicitly post-hoc six-test family at the same threshold (Table 6). All tests in both families pass; their largest unadjusted p-values are 0.000751 and 0.00794, respectively. ALFWorld's held-out family contains 20 comparisons (12 supervised pairwise contrasts and eight no-gate contrasts), with threshold 0.0025. The collection-task evaluation uses one GraphStitch–Hindsight comparison at 0.05. The two no-gate collection-task comparisons are each tested at 0.05 without correction across the pair. Approximate 80%-power detectable differences are 3–4pp for collection tasks and 6.6–8.3pp for held-out greedy, calculated at uncorrected $\alpha = 0.05$ conditional on the trained policies. These values are not confidence intervals or upper bounds on the true effect, and do not account for the stricter held-out family threshold or training-seed variability.

Successful-trajectory interpretation. The union-based construction unions selected single-episode demonstrations with composed paths; the Countdown sets are independently extracted (A.3). The following standard bound explains an idealized successful-trajectory objective, conditional on valid labels.

Proposition 1 (standard successful-subset bound)**.** *For a deterministic task t, let W_t be its successful, terminated action sequences within a finite horizon. Termination is part of the sequence, so the represented events are disjoint. For any nonempty set of distinct sequences $C_t \subseteq W_t$,*

$$\log P_{\pi_\theta}(W_t \mid t) \;\geq\; \frac{1}{|C_t|} \sum_{\sigma \in C_t} \log P_{\pi_\theta}(\sigma \mid t) + \log |C_t|.$$

For fixed C_t, maximizing the average complete-sequence log likelihood maximizes this lower bound. This is a standard Jensen/variational interpretation of successful-trajectory imitation, not a new optimization result (Dayan & Hinton, 1997; Singh et al., 2024).

Proof and scope. Distinct terminated sequences are disjoint events, giving $P_\pi(W_t \mid t) \geq \sum_{\sigma \in C_t} P_\pi(\sigma \mid t)$. Jensen's inequality with uniform weights gives the bound. Arbitrary unfinished prefixes need not be disjoint, and invalid labels do not belong to W_t (A.4). Actual assistant-token losses use environment-specific packing and normalization, so the expression is an idealized interpretation, not an identity for the implemented loss. The bound concerns assigned goals; transfer to evaluation goals requires additional assumptions (Ghosh et al., 2021). Adding low-likelihood sequences can lower the uniform-subset bound. No improvement guarantee for separately trained policies follows.

A.7 Complete Blocksworld prompt example

The following is a complete prompt for one collection task, including the system message, the two fixed few-shot exchanges, the task-specific user message, and the assistant generation prefix. Role and thinking delimiters are shown as rendered by the shared chat template. Other tasks replace the final GOAL, INITIAL STATE, and ACTIONS fields; the behavioral probe additionally replaces the goal. The action strings in the examples and final user message are retained alongside the system message's numbered-action instructions, exactly as in the evaluated configuration.

```
<|im_start|>system
I am playing with a set of blocks where I need to arrange the blocks into stacks.
↪  Here are the actions I can do

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.
I can only pick up a block if the block is on the table and the block is clear. A
↪  block is clear if the block has no other blocks on top of it and if the block
↪  is not picked up.
I can only unstack a block from on top of another block if the block I am
↪  unstacking was really on top of the other block.
```

```
I can only unstack a block from on top of another block if the block I am
↪  unstacking is clear.
Once I pick up or unstack a block, I am holding the block.
I can only put down a block that I am holding.
I can only stack a block on top of another block if I am holding the block being
↪  stacked.
I can only stack a block on top of another block if the block onto which I am
↪  stacking the block is clear.
Once I put down or stack a block, my hand becomes empty.
Once you stack a block on top of a second block, the second block is no longer
↪  clear.

You will be given the goal, the initial state, and a numbered list of every action
↪  in this problem. Write one complete plan that transforms the initial state
↪  into a state satisfying every GOAL literal. The plan is executed exactly as
↪  written, action by action, with no further input from you.

OUTPUT CONTRACT
- Reply with the plan as action numbers from ACTIONS, separated by single spaces,
↪  ending with STOP.
- Example of the format: 12 7 19 STOP
- Numbers and STOP only; no words, no punctuation, no explanations.<|im_end|>
<|im_start|>user
GOAL
- on(c,b)

INITIAL STATE
- clear(b)
- clear(c)
- handempty()
- on(b,a)
- ontable(a)
- ontable(c)

ACTIONS
- pick-up(a)
- pick-up(b)
- pick-up(c)
- put-down(a)
- put-down(b)
- put-down(c)
- stack(a,b)
- stack(a,c)
- stack(b,a)
- stack(b,c)
- stack(c,a)
- stack(c,b)
- unstack(a,b)
- unstack(a,c)
- unstack(b,a)
- unstack(b,c)
- unstack(c,a)
- unstack(c,b)

Reply with the complete plan as space-separated actions ending with STOP.<|im_end|>
<|im_start|>assistant
pick-up(c) stack(c,b) STOP<|im_end|>
<|im_start|>user
GOAL
- on(b,a)

INITIAL STATE
- clear(a)
- handempty()
- on(a,b)
- on(b,c)
- ontable(c)

ACTIONS
```

```
- pick-up(a)
- pick-up(b)
- pick-up(c)
- put-down(a)
- put-down(b)
- put-down(c)
- stack(a,b)
- stack(a,c)
- stack(b,a)
- stack(b,c)
- stack(c,a)
- stack(c,b)
- unstack(a,b)
- unstack(a,c)
- unstack(b,a)
- unstack(b,c)
- unstack(c,a)
- unstack(c,b)

Reply with the complete plan as space-separated actions ending with STOP.<|im_end|>
<|im_start|>assistant
unstack(a,b) put-down(a) unstack(b,c) stack(b,a) STOP<|im_end|>
<|im_start|>user
GOAL
- on(b,d)
- on(c,b)

INITIAL STATE
- clear(a)
- clear(d)
- handempty()
- on(b,c)
- on(d,b)
- ontable(a)
- ontable(c)

ACTIONS
- pick-up(a)
- pick-up(b)
- pick-up(c)
- pick-up(d)
- put-down(a)
- put-down(b)
- put-down(c)
- put-down(d)
- stack(a,b)
- stack(a,c)
- stack(a,d)
- stack(b,a)
- stack(b,c)
- stack(b,d)
- stack(c,a)
- stack(c,b)
- stack(c,d)
- stack(d,a)
- stack(d,b)
- stack(d,c)
- unstack(a,b)
- unstack(a,c)
- unstack(a,d)
- unstack(b,a)
- unstack(b,c)
- unstack(b,d)
- unstack(c,a)
- unstack(c,b)
- unstack(c,d)
- unstack(d,a)
- unstack(d,b)
- unstack(d,c)
```

```
Reply with the complete plan as space-separated actions ending with STOP.<|im_end|>
<|im_start|>assistant
<think>

</think>
```