COMPOSITIONAL LANGUAGE-INSTRUCTED REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

A compositional natural-language instruction decomposes a sparse-reward task into an ordered sequence of subgoal instructions, without providing a criterion for when to advance. Most prior approaches learn a completion signal and deliver it to the agent as auxiliary reward. We present GLEAN (Grounded Language-Effect ANchoring), which learns a completion detector from segmented offline demonstrations and uses it to advance the policy through the subgoals. The detector summarizes the egocentric observations so far into a filter state, which is also an input to the policy. It scores the change in that state since the previous completed subgoal against the active subgoal instruction and compares the score with a threshold calibrated on held-out demonstrations. The policy is initialized by behavior cloning and, throughout online reinforcement learning, continues imitating the demonstrations' subgoal segments while each detection advances it to the next subgoal instruction; we call this grounded subgoal anchoring. On six BabyAI levels with a matched interaction budget, GLEAN achieves higher final success than every baseline that receives a completion signal as reward on all five compositional levels, including baselines that start from a policy behavior-cloned on the same demonstrations. Providing the environment's exact completion predicates as reward still yields lower final success than GLEAN with its learned detector, so the advantage of advancement with continued imitation over reward does not come from detector errors.

1 INTRODUCTION

Many tasks that people want agents to perform are specified in natural language as an ordered list of steps, such as "go to the box, then go to a ball" in a grid world (Chevalier-Boisvert et al., 2019), a household routine (Shridhar et al., 2020), or a manipulation sequence (Mees et al., 2022). Language is how people already describe such tasks, and writing an instruction is easier than designing a reward function for every task (Luketina et al., 2019). We study *language-instructed reinforcement learning* (RL) with such compositional instructions: the agent receives the instruction and partial observations of its surroundings, and the environment rewards it only when the whole instruction is completed. Learning from this sparse reward is hard because several behaviors must be performed in order before any return (Nair et al., 2018), and until the agent first succeeds, policy-gradient methods receive almost no information about which actions help (Kakade & Langford, 2002).

A compositional instruction divides the task into *subgoals*, one per step, that must be completed in order. The instruction says what to do at each stage but not when a stage is done. A policy conditioned on the whole instruction must infer from its observations which subgoal it is currently pursuing (Zhang et al., 2026). Conditioning the policy on one subgoal instruction at a time, the *active* subgoal, removes this inference, but the agent must then decide when the active subgoal is complete and advance to the next. Completion is a property of the environment's state, which the agent observes only partially, and outside of simulation no completion signal is available. Errors in this decision are costly: a premature advancement skips a subgoal for the rest of the episode, a missed completion leaves the agent pursuing a subgoal it has already achieved, and under sparse reward neither error is signaled before the episode ends.

Existing language-instructed RL methods mostly learn a completion or language–state compatibility signal and provide it as reward (Mirchandani et al., 2021; Du et al., 2023; Nair et al., 2022). Such

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.

a reward is most informative near the completion of a subgoal, so it gives little guidance toward a subgoal the agent cannot yet reach; a learned signal can also be exploited through its false positives. Other methods change which part of the instruction conditions the policy, selected by a learned high-level policy (Zhang et al., 2026) or removed when a learned similarity score rises sharply (Saghafian et al., 2025), and neither makes a completion decision with controlled errors. Imitation learning is a separate line of work that learns from whole-task demonstrations, by behavior cloning (BC), supervised learning of the demonstrated actions, or by adding demonstrations to online RL (Hester et al., 2018; Rajeswaran et al., 2018; Ball et al., 2023). BC alone makes errors that compound once the agent leaves the demonstrated states (Ross et al., 2011), and online RL can correct them but can also erode the cloned behaviors (Wołczyk et al., 2024).

We consider language-instructed RL in which the agent also has demonstrations segmented by subgoal, each segment labeled with the subgoal instruction it completes. Such segmentation is often available: a scripted expert that completes one subgoal at a time produces it by construction, as in BabyAI, and instruction corpora such as ALFRED provide subgoal-annotated demonstrations (Shridhar et al., 2020). Segmented demonstrations can supervise both what the agent should do for each subgoal and when each subgoal is complete.

We propose GLEAN (Grounded Language-Effect ANchoring), which uses a learned completion signal to decide which subgoal the policy is conditioned on, rather than providing it as reward, so the active subgoal's instruction guides the policy throughout the subgoal and the policy gains nothing from triggering the signal's errors. A completion detector, learned from the segments and calibrated so that premature advancements occur at a chosen rate, advances the policy through the subgoals. The policy is initialized by BC on the segments and, during online RL, continues imitating them to mitigate forgetting, which we call *grounded subgoal anchoring*.

On six BabyAI levels with a matched interaction budget, GLEAN outperforms every baseline that receives a completion signal as reward on all compositional levels, including baselines that start from the same demonstrations and baselines rewarded by the environment's own completion predicates, which read the true state. Our contributions are threefold. First, we formulate language-instructed RL with subgoal-segmented demonstrations and show, with the completion signal held fixed, that advancing the policy through the subgoals outperforms providing the signal as reward when completions are detected reliably, and that detection errors are the bottleneck on the longest subgoal chains. Second, we introduce a completion detector for partial observability whose thresholds are calibrated by split-conformal prediction to a target false-advancement rate and whose internal state also informs the policy. Third, we combine demonstrations with online RL at the level of subgoals, imitating each segment under its own subgoal instruction, the same conditioning the detector provides online, and show that this online subgoal imitation matters, most on the longest chains.

2 Related Work

Language-assisted reinforcement learning. Natural language has been widely used to improve the sample efficiency of RL (Luketina et al., 2019). A common form of assistance derives reward from language. Du et al. (2023) proposed ELLM, which pretrains an agent without task reward by rewarding it for achieving goals suggested by a large language model. Other methods reward vision–language similarity (Fan et al., 2022; Rocamonde et al., 2024) or train a classifier that recognizes goal states for an instruction from expert examples (Bahdanau et al., 2019). Mirchandani et al. (2021) proposed ELLA, which pairs an offline termination classifier for low-level instructions with an online relevance classifier and adds a bonus when a relevant low-level instruction terminates.

Compositional instruction following. Several methods describe a task as language subtasks or pieces of guidance and condition the policy on a part of this description that changes during the episode, chosen by a learned high-level policy or updated when a subtask is detected as complete. Zhang et al. (2026) proposed HRLLI, a hierarchical method in which a high-level policy repeatedly selects the instruction tip most relevant to the current state and a low-level policy acts on it; the tips form an unordered set that need not describe a complete step-by-step procedure, and no tip is marked as completed. Sharma et al. (2022) proposed $(SL)^3$, which induces the segmentation of demonstrations into skills when only 10% of the demonstrations carry language annotations. Saghafian et al. (2025) proposed CAREL, which splits the instruction into subtasks by rules, adds

a contrastive auxiliary loss from video–text retrieval on successful episodes, and removes a subtask from the instruction that conditions the policy when its similarity score rises sharply. Reward machines represent sequential tasks as automata whose transitions are triggered by detected events, and Li et al. (2024) show that tracking the automaton state from noisy event detectors, by hard decisions or by independent probabilistic updates, can mislead the policy.

Success detection. Deciding whether an instruction has been carried out can itself be learned. Nair et al. (2022) proposed LOReL, which learns a binary classifier over state changes from crowd-annotated offline data and maximizes it as a reward under model-predictive control. Ren et al. (2023) proposed KnowNo, which brings split-conformal calibration to embodied language decisions, calibrating a planner's uncertainty so that the robot asks for help with a coverage guarantee.

Learning from demonstrations and online interaction. Many methods combine offline expert demonstrations with online RL. The simplest initializes the policy by BC and fine-tunes it with RL (Rajeswaran et al., 2018), which can erode the cloned behaviors (Wołczyk et al., 2024). Other methods keep the demonstrations throughout online training. Deep Q-learning from demonstrations keeps a supervised loss on them for the whole run (Hester et al., 2018), Nair et al. (2018) apply a BC loss only where the critic rates the demonstrated action above the policy's, demonstration-augmented policy gradient anneals a demonstration term in the policy gradient (Rajeswaran et al., 2018), and kickstarting distills a teacher policy into the learner (Schmitt et al., 2018). Offline-to-online methods pretrain on the prior data by offline RL and continue training online (Nair et al., 2020; Kostrikov et al., 2022; Nakamoto et al., 2023), and RL with prior data draws half of every off-policy batch from the prior data without an imitation term (Ball et al., 2023). Self-imitation learning imitates the agent's own successful episodes (Oh et al., 2018), and Cideron et al. (2020) proposed HIGhER, which extends hindsight relabeling (Andrychowicz et al., 2017) from goal states to instructions by learning an instruction generator from the agent's own successes and relabeling its failures. These methods use demonstrations or experience of the whole task. Hierarchical imitation and RL (Le et al., 2018) instead imitates an expert's subgoal choices interactively at the high level while learning each subgoal's low-level policy by RL.

3 Problem Setting

We consider reinforcement learning (RL) in an instruction-augmented partially observable Markov decision process (POMDP), defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, P, Z, R, \mathcal{I}, \gamma)$, where $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ denote the state, action, and observation spaces, respectively; $P(s'|s,a)$ denotes the transition model; $Z(o|s,a)$ denotes the observation model; R is the reward function; $\gamma \in [0,1)$ is the discount factor; and $\mathcal{I}$ denotes a set of natural-language instruction sequences. Each RL task is assumed to be provided with a natural-language instruction sequence $\mathcal{I} \ni I = (I_1, \ldots, I_K)$, where each atomic instruction I_i describes the i-th subtask needed to be accomplished as a *subgoal* in a prescribed sequential order toward completing the overall task *mission*.

To facilitate grounding between language instructions and agent behavior, we assume access to a small instruction-annotated dataset $\mathcal{D} = \{(\tau_j, I_j)\}_{j=1}^{N}$, where each trajectory segment $\tau_j = (o_1, a_1, \ldots, o_{T_j})$ is annotated with an atomic instruction I_j describing the corresponding subtask (Shridhar et al., 2020).

Under this instruction-augmented POMDP formulation, the agent maintains a recurrent representation of its observation history, $h_t = f_{\text{enc}}(h_{t-1}, o_t)$, and learns a language-conditioned policy $\pi_\theta(a_t|h_t, I)$, with the objective of maximizing the expected discounted return, $\max_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$.

4 Method

We propose GLEAN (Grounded Language-Effect ANchoring) for effective RL with compositional language instructions under the problem setting formulated in Section 3. GLEAN trains in three stages (Figure 1). First, a subgoal completion detector is learned from the annotated offline segments in $\mathcal{D}$, supporting subsequent compositional language-instructed online learning. Second, a detector-conditioned behavior cloning module is designed to enhance policy learning in a supervised manner

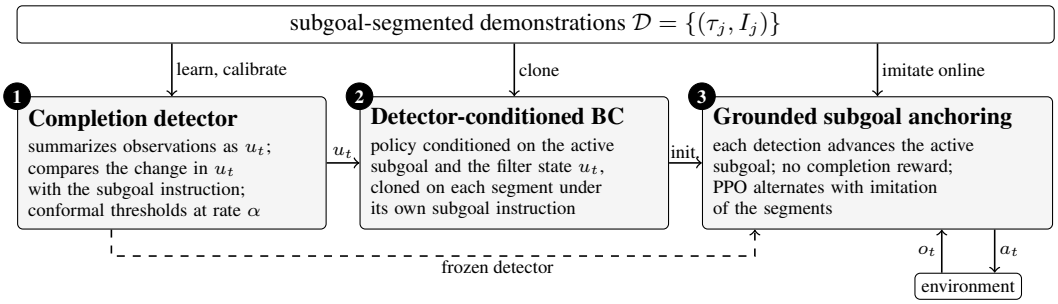

Figure 1: Overview of GLEAN. A completion detector (1) and a policy (2) are learned offline from the same subgoal-segmented demonstrations. Online (3), each detection advances the policy to the next subgoal instead of providing reward, while the policy continues imitating the segments.

from the offline dataset $\mathcal{D}$. Third, the policy is trained online with proximal policy optimization (PPO), using the frozen detector to advance the language instruction guidance, while imitating the offline segments via behavior cloning loss minimization.

4.1 Grounding instructions in their effects

Matching an instruction against a single observation is unreliable under partial observability, since different executions of the same instruction begin and end in visually different partial views. The detector therefore grounds each instruction in the *effect* of its execution, the change it induces in a summary of the observation stream. A filter ϕ, a convolutional frame encoder followed by a causal transformer, summarizes the observations into the *filter state* $u_t = \phi(o_1, \ldots, o_t)$. The filter runs over the whole episode and is never reset. We write u_{ref} for the filter state at the previous subgoal completion, or the initial filter state before any completion. When the detector is trained offline, the completions that set u_{ref} are labels from the demonstration segmentation; online, during policy training and evaluation, they are the detector's own predictions. Because the filter state accumulates the whole episode, it also reflects the effects of earlier subgoals; its change since the last completion isolates what happened during the current subgoal, so the score is specific to the active instruction. A frozen sentence encoder embeds any instruction I as a vector e_I, and a learned linear projection g_η maps it into the detector's space as $z_I = g_\eta(e_I)$. The score compares the effect $u_t - u_{\mathrm{ref}}$ with the projected embedding,

$$d_t(I) = \cos\big(W_r(u_t - u_{\mathrm{ref}}),\, z_I\big), \tag{1}$$

with W_r a learned affine map, a linear layer with bias.

The segments in $\mathcal{D}$ come from complete demonstration episodes, each divided at its subgoal completions, so consecutive segments of an episode meet at a completion step. The demonstration segmentation supplies all training labels. Every step of an episode is scored against a fixed vocabulary $\mathcal{V}$ of candidate texts, the subgoal instructions the task families can issue. A text is positive at the *completion step* of a demonstrated subgoal it matches, the first step of the following segment, or the final step for a terminal subgoal. Negatives pair the same texts with other steps, which teaches timing, and pair texts that match none of the episode's subgoals, which teaches discrimination. Unlabeled uniform-random rollouts join as purely negative episodes, and a text's steps after its own completion are left out. The loss is a binary cross-entropy over these (step, text) pairs,

$$\mathcal{L}_{\mathrm{det}} = \mathbb{E}_{(t,v)\sim\mathcal{P}\cup\mathcal{N}}\big[\mathrm{BCE}\big(\sigma(d_t(v)/\beta),\, \mathbf{1}[(t,v)\in\mathcal{P}]\big)\big], \tag{2}$$

with σ the logistic sigmoid, $\mathbf{1}[\cdot]$ the indicator function, $\mathcal{P}$ and $\mathcal{N}$ the positive and negative pairs, and β a fixed training-only temperature. All detector parameters are then frozen, so the completion criterion stays fixed throughout online training.

4.2 Calibrated advancement

A premature advancement skips its subgoal for the rest of the episode and goes unsignaled under sparse reward, so before any online interaction the detector's thresholds are calibrated to control *false advancement*, an advancement for a subgoal that has not been completed. A detection, a score

reaching its threshold, advances the subgoal online. In offline training a text is scored only until its own completion, but online a subgoal that is never completed is scored until the episode ends, so calibration scores each text on held-out demonstration episodes from the previous subgoal's completion step onward. A *null* is the running maximum $d^{\max}(I) = \max_t d_t(I)$ over such a window for a text I that matches none of the episode's subgoals. Any detection of I there is a false advancement, and one occurs if and only if the null reaches the threshold. We collect nulls on a calibration split held apart from training and set the thresholds by split-conformal calibration (Papadopoulos et al., 2002; Lei et al., 2018). Because null distributions differ across task families ℓ, the sets of instructions produced by one generator, and subgoal kinds κ, the behavior a subgoal asks for, such as going to an object or picking it up, thresholds are calibrated within each cell (ℓ, κ) (Mondrian calibration; Vovk et al., 2003). With $m_{\ell,\kappa}$ nulls $d^{\max,(1)}, \ldots, d^{\max,(m_{\ell,\kappa})}$ in cell (ℓ, κ), the threshold at target false-advancement rate α is their empirical $(1-\alpha)$ quantile $\hat{Q}_{1-\alpha}$,

$$\epsilon_{\ell,\kappa} = \hat{Q}_{1-\alpha}\big(d^{\max,(1)}, \ldots, d^{\max,(m_{\ell,\kappa})}\big). \tag{3}$$

When online nulls follow the same distribution as the calibration nulls, this threshold holds the false-advancement rate of each cell near α. Online, the policy's episodes differ from the expert's, so the control is only approximate (Barber et al., 2023).

Online, an instruction controller holds the active subgoal index i, starting at 1, and conditions the policy on I_i. Subgoal $i < K$ is detected as complete at the first step t_i where its score reaches the threshold of its own cell, with $\kappa(i)$ the kind of subgoal i,

$$d_t(I_i) \;\geq\; \epsilon_{\ell,\kappa(i)}, \tag{4}$$

whereupon $i \leftarrow i{+}1$ and $u_{\mathrm{ref}} \leftarrow u_t$. Advancement is latched: subgoals advance in order and cannot recur. Calibration holds the false-advancement rate of each detection near α on windows of texts that match none of the episode's subgoals; we assume that a subgoal of the episode is no more likely to be detected before its completion than such a text, consistent with held-out uniform-random rollouts, on which the first detection precedes the true completion in 0.5% of episodes on PUTNEXTLOCAL and 2.7% on GOTOSEQ. A union bound then limits the probability that any advancement in the episode is premature to about $(K{-}1)\alpha$, even though successive detections are dependent, because subgoal $i{+}1$ is scored from subgoal i's detection.

4.3 DETECTOR-CONDITIONED BEHAVIOR CLONING

Detection says when a subgoal is complete, not how to accomplish it, so GLEAN clones the policy on the demonstration segments with two inputs that flat BC, which is conditioned on the whole instruction, lacks: the active subgoal's instruction in place of the whole instruction, and the detector's filter state. The recurrent policy $\pi_\theta(a_t \mid h_t, e_{I_i}, u_t)$ receives its own recurrent history h_t, the embedding e_{I_i} of the active subgoal instruction, and the filter state u_t through a linear adapter. Conditioning on the active subgoal lets the policy learn per-subgoal behaviors that recur across tasks. The filter state gives the policy a representation shaped by the detector's loss in equation 2 to reflect the changes that complete instructions, supervision that the policy's own encoder does not receive from cloning actions alone, and it lets the policy act on the same summary of the observation history on which the controller decides advancement. Unlike the score, the policy receives the filter state itself, which retains the effects of earlier subgoals, such as an object already moved, and does not depend on u_{ref}, so a premature advancement does not alter it.

For cloning, the filter runs over each demonstration episode from its first observation, so at every step of a segment the policy receives the filter state it would receive online at that point of the episode. The detector-conditioned BC loss is the negative log-likelihood of the recorded actions,

$$\mathcal{L}_{\mathrm{BC}}(\theta; \mathcal{D}) = -\mathbb{E}_{(t,j)\sim\mathcal{D}}\big[\log \pi_\theta\big(a_t \mid h_t,\, e_{I_j},\, u_t\big)\big], \tag{5}$$

with (t, j) a step t of segment (τ_j, I_j) drawn uniformly from all segment steps in the dataset $\mathcal{D}$ of Section 3. Flat BC minimizes the same loss conditioned on the whole instruction and without the filter state. GLEAN initializes the whole policy, including perception, memory, and action head, by minimizing equation 5 over minibatches of segments, and keeps $\mathcal{D}$ for the online stage.

4.4 ONLINE LEARNING OBJECTIVES

Online, the policy acts under the instruction controller: at each step it is conditioned on the active subgoal I_{i_t}, with i_t set by the advancement rule in equation 4, and on the filter state u_t. The

detector's decisions change only which subgoal conditions the policy and never enter the reward. The reinforcement objective is the expected discounted return of Section 3 under the environment's sparse reward, with the policy acting under the controller,

$$J(\theta) = \mathbb{E}_{a_t \sim \pi_\theta(\cdot|h_t, e_{I_{i_t}}, u_t)} \Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \Big]. \tag{6}$$

Maximizing J trains the policy on the states it visits itself, where the errors of a cloned policy compound (Ross et al., 2011), but the sparse reward gives little signal there and online updates can erode the cloned behaviors (Wołczyk et al., 2024), while the BC loss in equation 5 preserves them but gives no signal outside the demonstrations. Following methods that add an imitation term on demonstrations to the RL objective (Hester et al., 2018; Rajeswaran et al., 2018), GLEAN trains the policy for the *grounded subgoal anchoring* objective

$$\min_\theta \; \mathcal{L}_{\mathrm{GSA}}(\theta) = -J(\theta) + \lambda \, \mathcal{L}_{\mathrm{BC}}(\theta; \mathcal{D}), \tag{7}$$

whose two terms draw on different data, J on the policy's own episodes under the controller and $\mathcal{L}_{\mathrm{BC}}$ on the fixed demonstration segments, with $\lambda > 0$ trading them off.

GLEAN alternates the two terms rather than descending their weighted sum, following the update schedule of self-imitation learning (Oh et al., 2018), which takes a fixed number of imitation steps on minibatches from a separate buffer after each on-policy actor-critic update; unlike self-imitation learning, GLEAN imitates the expert's segments. Each training phase collects rollouts under the controller for a fixed number of environment steps and updates the policy with clipped PPO (Schulman et al., 2017), which ascends J, and then takes N_{im} gradient steps on $\mathcal{L}_{\mathrm{BC}}$ over minibatches of segments from $\mathcal{D}$, which we call *online imitation*. Because PPO clips its probability ratio against the policy that collected the current rollouts, imitating only between collection phases leaves every PPO update unchanged, and the next phase collects fresh rollouts, so the ratio is never computed against a policy that imitation has since changed. The number N_{im} takes the role of λ: it sets how strongly imitation counteracts the reinforcement updates, and $N_{\mathrm{im}} = 0$ removes online imitation. Because cloning fits no value function, a *critic warm-up* first trains PPO's value function with the policy frozen.

5 Experiments

5.1 Experimental settings

Benchmark environment. We evaluate on BabyAI (Chevalier-Boisvert et al., 2019), a procedurally generated gridworld benchmark for instruction following in which the agent receives a partial egocentric view. BabyAI supplies the completion predicates that the oracle controls require and is light enough for five-seed runs of many methods and ablation variants on each level. It provides instructions in a compositional synthetic language and a scripted expert that generates demonstrations on demand. Each level is a task family ℓ in the sense of Section 4, and subgoals have four kinds, *goto*, *pickup*, *open*, and *putnext*. We use six levels, which differ in subgoal kinds, room layout, and chain length. GoToLocal has a single *goto* subgoal. PutNextLocal has two, picking up an object and placing it next to another, and PutNextS5N2 applies the same grammar across two rooms. GoToSeqS5R2 (hereafter GoToSeq) chains 1–4 *goto* subgoals across a 2×2 grid of 5×5 rooms. OpenTwoDoors has two *open* subgoals, and MoveTwoAcrossS5N2 (hereafter MoveTwoAcross) chains two such pick-and-place pairs, four subgoals, the longest chain the subgoal vocabulary supports. Each level has one corpus of segmented expert episodes: 20,000 episodes on GoToLocal, PutNextLocal, and PutNextS5N2, and 10,000 on the other three, so the corpora hold 20,000 to 40,000 subgoal segments.

Comparison methods. We compare GLEAN with five published methods trained without demonstrations for the policy: PPO (Schulman et al., 2017) on the sparse reward; ELLA (Mirchandani et al., 2021), which adds a bonus when a termination classifier, trained on demonstrations, detects completion of a low-level instruction; HRLLI (Zhang et al., 2026), which selects subgoal instructions with a learned high-level policy; HIGhER (Cideron et al., 2020), which relabels failed episodes with instructions from a learned generator; and CAREL (Saghafian et al., 2025), which removes completed subtasks from the instruction when a learned similarity score exceeds twice its running

Table 1: **Main comparison at a matched 1M-step budget**: final success, mean±s.d. Sections: with demonstrations; without demonstrations; with the environment's completion predicates (oracle). Bold: best non-oracle mean per level; underline: second best.

method	GoToLocal	PutNextLocal	GoToSeq	OpenTwoDoors	PutNextS5N2	MoveTwoAcross
GLEAN (ours)	.996±.001	**.973±.011**	.891±.014	**.996±.005**	**.995±.006**	.469±.147
flat BC	.965±.012	.405±.068	.834±.007	.512±.012	.570±.028	.435±.044
subgoal BC w/ detector advancement	.971±.008	.739±.050	.713±.013	.212±.098	.559±.179	.043±.022
flat BC+PPO	.994±.004	.223±.176	.590±.193	.977±.016	.920±.039	.000±.000
flat BC+PPO+detector reward	.994±.004	.297±.137	.467±.112	.636±.375	.590±.151	.000±.000
HIGhER+BC	**.998±.002**	.710±.102	**.917±.012**	.707±.240	.950±.023	**.522±.095**
flat PPO	.815±.085	.000±.000	.027±.016	.000±.000	.000±.000	.000±.000
ELLA	.865±.081	.000±.000	.023±.010	.000±.000	.000±.000	.000±.000
HRLLI	.845±.108	.000±.000	.032±.010	.000±.000	.000±.000	.000±.000
HIGhER	.608±.242	.001±.002	.048±.010	.000±.000	.003±.003	.000±.000
CAREL	.700±.177	.000±.000	.034±.026	.000±.000	.000±.000	.000±.000
GLEAN w/ oracle advancement	.996±.004	.977±.008	.988±.003	1.000±.000	.997±.002	.994±.005
flat BC+PPO+oracle reward	.994±.004	.207±.126	.505±.125	.804±.161	.691±.112	.009±.009

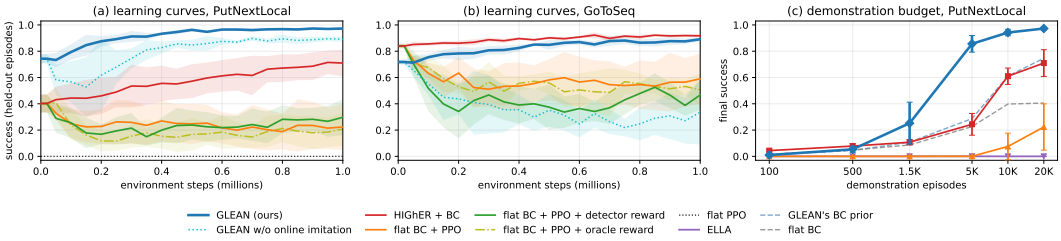

Figure 2: Mean±s.d. over five seeds. (a, b) Learning curves on PutNextLocal and GoToSeq, evaluated every 50K steps on 256 held-out episodes (final point on 512). Step 0 is the behavior-cloned policy; flat PPO starts from a random policy. (c) Final success on PutNextLocal against the number of demonstration episodes; dashed: BC priors before online training.

mean. The baselines with demonstrations start from a *flat* behavior-cloned prior (Pomerleau, 1988), conditioned on the whole instruction rather than on the active subgoal. Flat BC evaluates this prior without online training. Two baselines learn from both the demonstrations and online interaction: flat BC+PPO fine-tunes the prior with PPO on the sparse reward (Rajeswaran et al., 2018), and HIGhER+BC combines HIGhER with the prior and, during online training, keeps imitating the demonstrations (Hester et al., 2018) together with the agent's own successful episodes (Oh et al., 2018). Subgoal BC w/ detector advancement evaluates GLEAN's cloned prior without the filter-state input and without online training. Two signal-use controls provide a completion signal to the flat prior as reward: flat BC+PPO+detector reward rewards the learned detector's advancements, and flat BC+PPO+oracle reward rewards the environment's completion predicates. GLEAN w/ oracle advancement advances subgoals on the environment's completion predicates, which bounds what perfect completion information provides. The oracle variant and the oracle-reward control are the only runs that read the simulator state. Elsewhere, the completion predicates serve only to mark segment boundaries when the demonstrations are generated and to measure detection errors offline.

Implementation details. Every online method uses the same policy architecture and a matched budget of 10^6 environment steps. The policy reads an egocentric $7\times7\times3$ symbolic view through a convolutional encoder with feature-wise linear modulation (FiLM; Perez et al., 2018) and a 128-unit long short-term memory (LSTM), receives the instruction as a frozen 384-d sentence embedding, and is trained with recurrent PPO at entropy coefficient 0.005. The detector is a two-layer causal transformer over per-frame convolutional tokens, calibrated at $\alpha = 0.05$. Because the levels share one subgoal vocabulary, each detector is trained jointly on several levels and serves every online run on those levels that uses a detector, including the detector-reward control. GLEAN alternates 10k environment steps of PPO with $N_{\text{im}} = 64$ online-imitation steps on batches of 32 segments, after a 20k-step critic warm-up counted against the budget. We report the final checkpoint of each run, not its best, with success averaged over n=512 fixed held-out episodes, as the mean and population standard deviation over five seeds. The remaining settings are listed in the appendix.

Table 2: Ablations at the full corpus: final success, mean±s.d. over five seeds. Each row below GLEAN names its change to GLEAN. Top: component removals and the completion reward. Middle: imitation data. Bottom: policy input. CR: completion reward; FS: filter-state.

variant	GoToLocal	PutNextLocal	GoToSeq	OpenTwoDoors	PutNextS5N2	MoveTwoAcross
GLEAN	.996±.001	.973±.011	.891±.014	.996±.005	.995±.006	.469±.147
w/o BC init	.993±.003	.322±.173	.848±.021	.889±.205	.773±.201	.021±.017
w/o online imitation	.990±.004	.892±.023	.336±.246	.870±.247	.993±.004	.000±.000
w/o critic warm-up	.995±.002	.973±.003	.880±.029	.999±.002	.993±.003	.424±.139
+ CR	.995±.002	.955±.022	.874±.018	.991±.011	.985±.004	.500±.125
w/o advancement or FS input, + CR	.996±.001	.836±.015	.894±.013	.687±.236	.955±.021	.784±.098
+ self-imitation	.996±.003	.968±.016	.889±.021	.911±.105	.995±.003	.105±.046
w/o FS input	.995±.004	.927±.062	.817±.036	.998±.004	.992±.012	.248±.129
w/o observation input	.651±.038	.007±.002	.350±.051	1.000±.000	.048±.020	.017±.020
w/ filter-change input	.995±.001	.968±.005	.871±.018	1.000±.000	.996±.001	.101±.079

5.2 Main results

Table 1 reports final success on all six levels. GLEAN leads every baseline on PutNextLocal, OpenTwoDoors, and PutNextS5N2, and trails HIGhER+BC on GoToSeq and, within one standard deviation across seeds, on MoveTwoAcross. With oracle advancement GLEAN reaches 0.988 on GoToSeq and 0.994 on MoveTwoAcross, so completion detection limits both levels. Its largest margin is on PutNextLocal, where the closest baseline, subgoal BC w/ detector advancement, ends 0.23 lower. On single-subgoal GoToLocal, every method with demonstrations exceeds 0.96. Without demonstrations, no baseline exceeds 0.05 on any compositional level.

Advancement versus reward. Providing a completion signal to the flat prior as reward does not help on the compositional levels, whether the signal comes from the learned detector or from the environment's completion predicates. On PutNextLocal and GoToSeq, both signal-use controls underperform flat BC, the prior they start from, and on the other three compositional levels they underperform flat BC+PPO or stay near zero. GLEAN, which advances on the learned detector and keeps imitating the demonstrations, outperforms both controls on every compositional level, including the control that receives the oracle signal. Advancement without continued imitation does not account for the whole difference: GLEAN w/o online imitation ends far above the detector-reward control on PutNextLocal but below it on GoToSeq (0.336 against 0.467).

Continued imitation. The learning curves (Figure 2a, b) separate the variants by whether they keep imitating during reinforcement learning. GLEAN and HIGhER+BC, which do, rise from their cloned start on both levels. Every flat-prior variant that stops imitating reaches lower final success than its cloned start, with or without a completion reward. GLEAN w/o online imitation still rises on PutNextLocal but falls on GoToSeq. This pattern is consistent with forgetting of a behavior-cloned prior under online updates (Wołczyk et al., 2024).

5.3 Ablation Study

The ablations test which components GLEAN needs, whether the detector's decisions also help as reward or as a source of imitation data, and in what form the detector's information should reach the policy, changing one component at a time except in variant (v). The removals test necessity: (i) w/o BC init; (ii) w/o online imitation; and (iii) w/o critic warm-up. The reward variant (iv) + completion reward adds a reward of 1 to the environment's reward at each detector-decided advancement. Variant (v) w/o advancement or filter-state input, + completion reward provides the same reward at each detector decision but never advances the policy's instruction: the policy, its BC prior, and the imitated demonstrations are conditioned on the whole instruction, without the filter-state input. It is the detector-reward control of Table 1 with online imitation of the whole-task demonstrations added, as in deep Q-learning from demonstrations (Hester et al., 2018). The imitation-data variant (vi) + self-imitation adds to the demonstrations the agent's own segments between consecutive advancements, labeled with the active subgoal. The policy-input variants test what the policy should receive from the detector: (vii) w/o filter-state input, (viii) w/o observation input, which keeps only the filter state, and (ix) w/ filter-change input, which supplies $u_t - u_{\text{ref}}$ in place of u_t.

The BC prior and online imitation are the components GLEAN needs (Table 2). Without BC initialization, final success falls on every compositional level, most on PUTNEXTLOCAL (0.322 against 0.973) and MOVETWOACROSS (0.021 against 0.469). Removing online imitation costs most on the longest chains, GOTOSEQ and MOVETWOACROSS, where final success falls to 0.336 and 0.000, and at most 0.13 on the other compositional levels. Removing the critic warm-up changes final success by at most 0.011 except on MOVETWOACROSS, where it is 0.045 lower, within one standard deviation across seeds. Adding the completion reward does not help: it lowers final success slightly on four of five compositional levels and changes it on MOVETWOACROSS within one standard deviation across seeds. Providing the detector's decisions as reward in place of advancement, with online imitation kept, outperforms GLEAN on MOVETWOACROSS (0.784 against 0.469), matches it on GOTOSEQ (0.894 against 0.891), and underperforms it on PUTNEXTLOCAL, OPENTWODOORS, and PUTNEXTS5N2 (0.836, 0.687, and 0.955 against 0.973, 0.996, and 0.995). Without online imitation, the same reward ends at 0.000 on MOVETWOACROSS (Table 1). Self-imitation does not help either: it changes final success by at most 0.005 on four levels and lowers it on OPENTWODOORS and the four-subgoal MOVETWOACROSS (0.105 against 0.469), where a longer chain gives premature advancements more chances to admit segments whose subgoal was not completed.

The filter-state input raises final success on PUTNEXTLOCAL, GOTOSEQ, and, most, MOVETWOACROSS (0.973 against 0.927, 0.891 against 0.817, and 0.469 against 0.248) and changes it by 0.003 or less on the other three levels. The policy still needs the observations: with the filter state alone, final success is below 0.05 on PUTNEXTLOCAL, PUTNEXTS5N2, and MOVETWOACROSS, although it reaches 1.000 on OPENTWODOORS. Supplying the filter change $u_t - u_{\text{ref}}$ in place of u_t stays within 0.020 of GLEAN except on MOVETWOACROSS, where final success falls to 0.101.

5.4 SENSITIVITY ANALYSIS

The sensitivity analyses test how the results depend on the amount of demonstration data and on design choices held fixed elsewhere. They use GOTOLOCAL, PUTNEXTLOCAL, and GOTOSEQ.

Demonstration budget. We run GLEAN and three baselines (HIGhER+BC, flat BC+PPO, and ELLA) with every data-dependent component rebuilt from a single corpus of 100, 500, 1,500, 5,000, 10,000, or all episodes: the BC prior, the online-imitation demonstrations, and the detector's training and calibration data. On PUTNEXTLOCAL, GLEAN trails HIGhER+BC at 100 and 500 episodes, leads every baseline from 1,500 episodes on, and from 5,000 episodes reaches higher final success than every baseline's full-corpus result (0.856 against at most 0.710). On GOTOSEQ, HIGhER+BC leads GLEAN by 0.026 to 0.039 at every budget, and both stay above flat BC+PPO and ELLA. On GOTOLOCAL, every method except ELLA reaches at least 0.954 from 500 episodes on.

Detector design and online hyperparameters. Final success is robust to most single changes of detector design or online hyperparameter. It drops when the detector scores the filter state itself in place of its change since the last completion, at a stricter target rate α=0.01 on PUTNEXTLOCAL, and with a quarter of the online-imitation batches on GOTOSEQ. Pooling thresholds across levels loses per-cell error control without changing final success.

6 CONCLUSION

We presented GLEAN, a method for compositional language-instructed RL that learns and calibrates a completion detector from segmented demonstrations before any interaction. The detector sets the policy's active instruction and supplies its filter state as a policy input, and the policy keeps imitating the demonstrations online. With the completion signal held fixed, agents that use it for advancement while imitating the demonstrations outperform agents that receive it as reward, for the learned detector and for the environment's own completion predicates alike. GLEAN leads every baseline on PUTNEXTLOCAL, OPENTWODOORS, and PUTNEXTS5N2, and on PUTNEXTLOCAL it exceeds every baseline's full-corpus result with a quarter of the demonstrations. In the ablations, the BC prior and online imitation are the components GLEAN needs, and neither an added completion reward nor self-imitation helps. With oracle advancement, GLEAN reaches 0.988 on GOTOSEQ and 0.994 on MOVETWOACROSS, so detection limits the longest chains, and improving it is the main direction for future work.

AI USE STATEMENT

We used generative AI tools to write and edit code, to design experiments, to generate figures and tables from logged results, and to edit the paper. We have not used generative AI tools to generate synthetic datasets, to develop the theoretical model or conceptual framework, to formulate mathematical claims, or to provide proof ingredients or write proofs. Dataset cleaning and qualitative data analysis are not applicable to this work. We have reviewed all AI-assisted work and verified its correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All experiments use public BabyAI levels (Farama `minigrid` v2.3.1) and demonstrations from the platform's scripted expert. The supplementary material contains the exact corpora and anonymized code for corpus generation, training, evaluation, and every table and figure. Algorithms 1–3 and Appendices A.2 and A.3 give the pseudocode, every architecture and hyperparameter, and each baseline's implementation, and Appendix A.2.3 fixes the held-out evaluation seeds.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 5048–5058, 2017.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Seyed Arian Hosseini, Pushmeet Kohli, and Edward Grefenstette. Learning to understand goal specifications by modelling reward. In *International Conference on Learning Representations (ICLR)*, 2019.

Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning (ICML)*, 2023.

Rina Foygel Barber, Emmanuel J. Candès, Aaditya Ramdas, and Ryan J. Tibshirani. Conformal prediction beyond exchangeability. *The Annals of Statistics*, 51(2):816–845, 2023. doi: 10.1214/23-AOS2276.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations (ICLR)*, 2019.

Geoffrey Cideron, Mathieu Seurin, Florian Strub, and Olivier Pietquin. HIGhER: Improving instruction following with hindsight generation for experience replay. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 225–232, 2020. doi: 10.1109/SSCI47803.2020.9308603.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In *International Conference on Machine Learning (ICML)*, 2023.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. MineDojo: Building open-ended embodied agents with internet-scale knowledge. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 35, pp. 18343–18362, 2022.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Z. Leibo, and Audrunas Gruslys. Deep Q-learning from demonstrations. In *AAAI Conference on Artificial Intelligence*, 2018.

David Yu-Tung Hui, Maxime Chevalier-Boisvert, Dzmitry Bahdanau, and Yoshua Bengio. BabyAI 1.1. *arXiv preprint arXiv:2007.12770*, 2020.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, pp. 267–274, 2002.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations (ICLR)*, 2022.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Hoang M. Le, Nan Jiang, Alekh Agarwal, Miroslav Dudík, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. In *International Conference on Machine Learning (ICML)*, pp. 2917–2926, 2018.

Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani, and Larry Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113 (523):1094–1111, 2018.

Andrew C. Li, Zizhao Chen, Toryn Q. Klassen, Pashootan Vaezipoor, Rodrigo Toro Icarte, and Sheila A. McIlraith. Reward machines for deep RL in noisy and uncertain environments. In *Advances in Neural Information Processing Systems*, 2024.

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters (RA-L)*, 2022.

Suvir Mirchandani, Siddharth Karamcheti, and Dorsa Sadigh. ELLA: Exploration through learned language abstraction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6292–6299, 2018. doi: 10.1109/ICRA.2018.8463162.

Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. AWAC: Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Suraj Nair, Eric Mitchell, Kevin Chen, Brian Ichter, Silvio Savarese, and Chelsea Finn. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning (CoRL)*, 2022.

Mitsuhiko Nakamoto, Yuexiang Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-ql: Calibrated offline rl pre-training for efficient online fine-tuning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *International Conference on Machine Learning (ICML)*, 2018.

Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alex Gammerman. Inductive confidence machines for regression. In *Machine Learning: ECML 2002*, volume 2430 of *Lecture Notes in Computer Science*, pp. 345–356. Springer, 2002. doi: 10.1007/3-540-36755-1_29.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. doi: 10.1609/aaai.v32i1.11671.

Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems (NIPS)*, 1988.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems (RSS)*, 2018.

Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. Robots that ask for help: Uncertainty alignment for large language model planners. In *Conference on Robot Learning (CoRL)*, 2023.

Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 15 of *Proceedings of Machine Learning Research*, pp. 627–635, 2011.

Armin Saghafian, Amirmohammad Izadi, Negin Hashemi Dijujin, and Mahdieh Soleymani Baghshah. CAREL: Instruction-guided reinforcement learning with cross-modal auxiliary objectives. *Transactions on Machine Learning Research*, 2025. arXiv:2411.19787.

Simon Schmitt, Jonathan J. Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M. Czarnecki, Joel Z. Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, and S. M. Ali Eslami. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Pratyusha Sharma, Antonio Torralba, and Jacob Andreas. Skill induction and planning with latent language. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Vladimir Vovk, David Lindsay, Ilia Nouretdinov, and Alex Gammerman. Mondrian confidence machine. On-line Compression Modelling Project, Working Paper 4, Royal Holloway, University of London, 2003.

Maciej Wołczyk, Bartłomiej Cupiał, Mateusz Ostaszewski, Michał Bortkiewicz, Michał Zając, Razvan Pascanu, Łukasz Kuciński, and Piotr Miłoś. Fine-tuning reinforcement learning models is secretly a forgetting mitigation problem. In *International Conference on Machine Learning (ICML)*, 2024.

Hanping Zhang, Adam Koziak, and Yuhong Guo. Select-to-Act: Hierarchical reinforcement learning via adaptive language guidance. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2026, Proceedings, Part III*, Lecture Notes in Computer Science, pp. 147–164. Springer, 2026. doi: 10.1007/978-3-032-37664-0_9.

Jesse Zhang, Karl Pertsch, Jiahui Zhang, and Joseph J. Lim. SPRINT: Scalable policy pre-training via language instruction relabeling. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

A Appendix

A.1 Full three-stage pseudocode

GLEAN runs in three stages. Stage 1 trains, calibrates, and freezes the completion detector (Algorithm 1); Stage 2 behavior-clones the entire recurrent policy on subgoal-segmented demonstrations (Algorithm 2); Stage 3 is the online loop of Algorithm 3. We write $o_{\leq t}$ for the raw observation history, which the policy summarizes into h_t through its own recurrent encoder, and u_t for the detector's filter state over the same observations. The environment state s appears only inside the environment, in reward, termination, and predicates, and no component of GLEAN reads it.

Algorithm 1 Stage 1: offline detector training and calibration

Input: segmented demonstrations $\mathcal{D} = \{(\tau_j, I_j)\}$; unlabeled uniform-random rollouts; frozen sentence encoder; filter ϕ (frame encoder and causal transformer), projection g_η, readout W_r; temperature β
Split episodes $80/20$ into training $\mathcal{D}_{\mathrm{tr}}$ and held-out calibration $\mathcal{D}_{\mathrm{cal}}$
for each training step **do**
    Run the filter, $u_t = \phi(o_1, \ldots, o_t)$; form the score matrix $d_t(v)$ over $\mathcal{V}$
    BCE of equation 2: positives at (completion step, matching texts); negatives 20:1 from other steps and texts matching none of the episode's subgoals; steps after a text's own completion masked; random rollouts all-negative
    Update ϕ, g_η, W_r; sentence encoder fixed
end for
Freeze all detector parameters
Calibrate: on each held-out episode, run the filter from the episode start without reset and score each text matching none of the episode's subgoals from the previous subgoal's completion step to the episode end; each such text contributes one null (its running-maximum score); set $\epsilon_{\ell,\kappa}$ by equation 3 at α=0.05 (below 50 nulls, the kind's threshold pooled across levels); for the self-imitation variants, also set the admission thresholds $\epsilon^{\mathrm{adm}}_{\ell,\kappa}$ at α_{adm}=0.01, floored at $\epsilon_{\ell,\kappa}$
Output: frozen detector; $\epsilon_{\ell,\kappa}$ (and $\epsilon^{\mathrm{adm}}_{\ell,\kappa}$)

Algorithm 2 Stage 2: offline detector-conditioned behavior cloning

Input: subgoal-segmented demonstrations $\mathcal{D}$, frozen sentence encoder and detector
Run the frozen detector's filter over every demonstration episode to obtain u_t; initialize the recurrent actor $\pi_\theta(a_t \mid h_t, e_{I_j}, u_t)$ and critic V_ω
for each BC update **do**
    Sample segments; run the recurrent policy over the padded sequence; minimize equation 5
end for
Keep the segment dataset $\mathcal{D}$ for online imitation
Output: BC-initialized π_θ (perception, memory, and action head all cloned); the critic stays random and is warm-started in Stage 3

A.2 Instantiation, hyperparameters, and protocol details

A.2.1 BabyAI instantiation

BabyAI instructions carry a structured grammar in which GoTo, Pickup, Open, and PutNext instructions compose through Before, After, and And. Flattening that grammar yields the atomic instructions. A put-next instruction spans two behaviors, so we split it further into a *pickup* subgoal, picking up the object to be moved, and a *putnext* subgoal, placing it; we treat this split as part of the decomposition Λ, so each subgoal names a single behavior. Completion predicates only segment the generated demonstrations and measure false advancement offline, except in the two oracle runs; the predicate for the *pickup* half of a split put-next instruction is whether the agent carries an object matching the description of the object to be moved, the condition under which BabyAI's own pick-up verifier succeeds. A frozen 384-d sentence encoder (22M parameters) encodes every instruction: the policy consumes the embedding through FiLM, and the detector projects it to 128 dimensions.

Algorithm 3 Stage 3: online policy update (grounded subgoal anchoring)

Given: frozen detector with thresholds $\epsilon_{\ell,\kappa}$; BC-initialized π_θ, V_ω; segment dataset $\mathcal{D}$
Warm up V_ω for 20k environment steps with π_θ fixed
while fewer than 10^6 environment steps **do**
  for each rollout of 256 steps in each of 8 parallel environments, until 10k steps **do**
    for each step t in each environment **do**
      At an episode start: parse the instruction into $(I_1, \ldots, I_K)$; $i \leftarrow 1$; $u_{\text{ref}} \leftarrow u_1$
      Advance the filter state u_t on o_t; score $d_t(I_i)$ (equation 1)
      if $i < K$ and $d_t(I_i) \geq \epsilon_{\ell,\kappa(i)}$ **then**
        $u_{\text{ref}} \leftarrow u_t$; $i \leftarrow i{+}1$
      end if
      Sample $a_t \sim \pi_\theta(\cdot \mid h_t, e_{I_i}, u_t)$; execute
    end for
    Clipped recurrent-PPO update on the environment reward (ascends equation 6)
  end for
  for $n = 1, \ldots, N_{\text{im}}$ ($N_{\text{im}} = 64$) **do**
    Sample 32 segments from $\mathcal{D}$; take a gradient step on equation 5
  end for
end while

Demonstrations come from a scripted expert that completes one subgoal at a time and plans by breadth-first search, so they are subgoal-segmented by construction. We selected OPENTWODOORS, PUTNEXTS5N2, and MOVETWOACROSS from the platform's level inventory by episode-length and subgoal-count statistics under the scripted expert. Detector training and calibration use the episodes with at least one non-empty detectable segment (19,470, 20,000, and 7,798 on GOTOLOCAL, PUTNEXTLOCAL, and GOTOSEQ), while behavior cloning (BC) and online imitation use the full corpus. Between 100 and 200 unlabeled uniform-random rollouts per level join as negatives, and the calibration nulls come from the held-out expert split.

A.2.2 ARCHITECTURE AND HYPERPARAMETERS

Every online method shares one policy stack. It reads the egocentric $7{\times}7{\times}3$ symbolic view, embeds each cell with three 8-d identifier tables, and applies two $3{\times}3$ convolutions of width 64 with FiLM conditioning from the 384-d text embedding. The resulting 128-d feature feeds a 128-unit LSTM actor–critic, trained with RECURRENTPPO from Stable-Baselines3 and sb3-contrib (8 environments, $n_{\text{steps}}{=}256$, batch 256, 4 epochs, learning rate $3{\times}10^{-4}$, $\gamma{=}0.99$, generalized advantage estimation 0.95, clip 0.2). We fixed the entropy coefficient at 0.005 for every variant after a paired comparison against 0, run before the final runs; every final run was then trained from scratch on the shared stack.

Each GLEAN online-imitation batch holds 32 demonstration segments. In the self-imitation variants, when subgoal i is detected, the segment since the previous detection, or since the episode start, enters an online buffer $\mathcal{B}_{\text{online}}$ labeled with I_i, and the final subgoal's segment enters on environment success. The online buffer is a first-in-first-out queue of 8,000 segments, a fraction $f_{\text{demo}}{=}0.5$ of each batch comes from the demonstrations once the buffer holds segments, and while it is empty the whole batch comes from the demonstrations. A segment is admitted only if its score at the detection step also clears a stricter admission threshold $\epsilon^{\text{adm}}_{\ell,\kappa}$, the empirical $(1 - \alpha_{\text{adm}})$ quantile of the same nulls with $\alpha_{\text{adm}} = 0.01$, floored at $\epsilon_{\ell,\kappa}$.

The detector is a separate network, and the two networks keep separate recurrent states and share only the frozen sentence encoder and the observations; the policy never sees the projected embedding z_I or the detector's scores. Both recurrent states are reset only at the start of an episode: an advancement changes the policy's instruction and sets $u_{\text{ref}} \leftarrow u_t$, and neither network's recurrent state is reset there. A subgoal that is never detected stays active until the episode ends. In behavior cloning and online imitation, each segment is replayed from a zero policy recurrent state, while the filter state u_t it receives comes from the detector's run over the whole demonstration episode. The detector's frame encoder has the shape of the policy's without FiLM, followed by two linear layers to width 128. A two-layer causal transformer (4 heads, feed-forward 256, pre-norm, sinusoidal po-

sitions clamped at position 512) runs over those tokens, the affine map W_r, a linear layer with bias, maps the filter-state change into the score space, and the linear $384 \to 128$ projection g_η produces z_I. The vocabulary $\mathcal{V}$ also contains paraphrases of each subgoal instruction without the color word, and a text matches a demonstrated subgoal when it names the same kind and object. Training subsamples negatives at 20:1 with temperature $\beta=0.1$ for 15 epochs. In equation 3, $\hat{Q}$ is the linearly interpolated empirical quantile, which lies within two ranks of the finite-sample-corrected conformal quantile, whose false-advancement rate is at most α, so when online nulls follow the same distribution as the calibration nulls, the false-advancement rate of each cell is at most $\alpha + 2/(m_{\ell,\kappa} + 1)$. A cell with fewer than 50 nulls uses its kind's threshold pooled across levels. Online, the detector runs incrementally in NumPy with a key–value cache. Each training window's reference is the preceding demonstrated completion, or the first step when none precedes, and steps after a text's own completion are excluded, because online advancement latches at the first crossing.

The online-imitation batch count was the most sensitive setting during development. At a small early corpus, 32 batches rose then decayed ($0.16 \to 0.12$) where 64 was still rising at the end of the budget. Re-weighting demonstration sampling had no effect.

A.2.3 Evaluation protocol

We evaluate on fixed held-out episode sets built from reset seeds $7{,}000{,}000+i$, with $n=256$ episodes for periodic evaluations and $n=512$ for the final one. The policy acts deterministically and we record the environment's binary success flag in place of BabyAI's length-discounted return. No shaping or completion reward is provided at evaluation, and subgoals advance as in training: by the detector, or by the completion predicates in the oracle variants. The population standard deviation divides by the number of seeds (`np.std`). Main-table and ablation results average five seeds. The detector-design, online-hyperparameter, and one-axis corpus-size analyses average three and vary an earlier configuration, GLEAN + self-imitation + completion reward w/o filter-state input.

The environment is the Farama `minigrid` port of BabyAI (v2.3.1). BabyAI 1.0 and 1.1 use different agent architectures and different demonstration-requirement protocols (a Gaussian-process estimate against a fixed grid of budgets), so their numbers are not directly comparable, and the BabyAI 1.0 baselines report maximum validation success, which is generous relative to final-checkpoint results. Every ablation and budget number comes from final runs on the shared policy architecture. The retention-mechanism study is an earlier study, reported at its own $n=200$ protocol. Budgets are stated in environment steps and demonstration counts in episodes, and the two are never interchanged.

A.2.4 Compute

We ran the experiments on seven lab machines: three with $4\times$, $4\times$, and $2\times$ RTX 2080 Ti (11 GB), one with $4\times$ RTX A5000 (24 GB), one with $4\times$ RTX 3060 (12 GB), and two with one RTX 4090 (24 GB) each. With three runs per RTX 2080 Ti, a 1M-step run takes 2–7 hours, with a median of 3; the longest runs, HIGhER and the large additional levels, take up to 18. The reported results come from approximately 1,300 million-step runs under the unified protocol, scheduled by a queueing dispatcher.

A.2.5 Data-efficiency accounting

Table 3 sets our demonstration budgets against the platform's published requirements on the same levels.

The literature marks PutNextLocal as needing up to about twice as many demonstrations as GoToLocal, so its difficulty is not an artifact of our pipeline. Our full corpus is two-fifths of the smallest budget at which BabyAI 1.1 reports 99% on GoToLocal, a fifth of the corresponding PutNextLocal budget, and one percent of the budget used on the unrestricted GoToSeq level. The published numbers are for full-policy imitation to 99%, a different question from completion detection, and we measure the detector's own data requirement directly in Table 5, where a 1,500-episode detector loses 0.013 and 0.018 against the unchanged detector over the same three seeds.

The online budget is small against published pure-RL costs as well. Converting the platform's episode-denominated pure-RL costs to frames under the bounds least favorable to us, published

Table 3: Demonstration budgets against published requirements (episode counts). Hui et al. (2020) values are for the revised architectures and give the smallest budget on their tested grid (5, 10, 50, 100, 500K) that reaches 99%. †Not reached: 1M demonstrations, the budget used there for every level, reach 96.7% on the unrestricted GoToSeq level, one seed.

	GoToLocal	PutNextLocal	GoToSeq
our full corpus	20,000	20,000	10,000
our budget sweep			100–10,000
unlabeled random rollouts	200	200	100
to 99% (Chevalier-Boisvert et al., 2019)	149–193K	245–323K	—
to 99% (Hui et al., 2020)	≤50K	≤100K	>1M†

Table 4: Related methods. ° = run head-to-head at matched budget on our levels; native benchmarks differ (ELLM Crafter/Housekeep, SPRINT (Zhang et al., 2024) ALFRED, HRLLI RTFM).

method	decomposition	completion / grounding model	policy use	demos
ELLM	LLM-suggested goals	embedding similarity (fixed)	once-per-goal reward	no
ELLA°	online relevance clf.	offline termination clf.	completion bonus	clf. only
HIGhER°	none (whole task)	generator from own successes	reward relabel	no
CAREL°	rule-based split	contrastive similarity spike	aux. loss + instr. masking	no
HRLLI°	selector over given tips	state–tip score; no completion	selection + reward	no
HER	none	achieved state	relabel → off-policy RL	no
SPRINT	human labels + LLM aggregation	segment end state; learned critic	offline RL pretrain + fine-tune	yes
GLEAN (ours)°	instruction grammar	frozen detector on obs. history	adv. + input + online imit.	yes

solve costs are at least 4.5M frames on GoToLocal and 27M on PutNextLocal (at least 1.2M and 10.8M for the lowest mean costs among the architectures compared in the BabyAI 1.1 report). No pure-RL cost is published for GoToSeq. Our entire budget of 1M frames, or environment steps, sits below the smallest converted cost, so low baseline results on the compositional levels are in line with published costs. On PutNextLocal, GLEAN reaches 0.942 with 10,000 demonstration episodes against 0.973 with the full corpus (Table 10); 10,000 is a tenth of the smallest demonstration count at which BabyAI 1.1 reports 99% imitation success on that level.

A.3 Baseline implementations, fidelity, and fairness

A.3.1 Method landscape

Table 4 places GLEAN among the methods that attach language or goals to reinforcement learning, and marks the ones we run head-to-head at a matched budget.

A.3.2 HRLLI

HRLLI pairs a learned REINFORCE selector with the shared executor. The selector picks one instruction from the candidate set every 10 steps and trains on the segment sum of a learned per-transition reward model plus a cosine alignment bonus with weight $\lambda_{\text{sel}}=0.1$, taken from an earlier version of the method (the published value is 0.01; the bonus acts only when an instruction has more than one candidate). The executor is the shared RecurrentPPO stack trained on environment reward plus the reward model's score ($\lambda_{\text{act}}=0.1$). At evaluation, selection is greedy and no shaped reward is provided. The candidate set is the instruction's decomposition before the put-next split, so nothing privileged enters.

On GoToLocal, PutNextLocal, and PutNextS5N2 the decomposition yields one candidate per instruction, so the selector has no choice and its policy-gradient loss is zero. OpenTwoDoors and MoveTwoAcross yield two candidates per instruction and GoToSeq 1–4, so only these three levels exercise selection.

A.3.3 HIGhER

Published configuration. The main-table row adapts the published configuration to our shared policy architecture. An instruction-conditioned RecurrentPPO policy trains from scratch, and a recurrent token-decoder instruction generator trains online on the agent's own successes with teacher

forcing and decodes greedily; a unit test confirms that it decodes held-out slot combinations. The trust gate has the form of their Algorithm 1, a minimum validation score and validation-set size. We open it at 200 successes and held-out exact-match 0.18; the published run opened it after 1,000 successes at about 0.18 accuracy. Relabels are delivered through self-imitation, the nearest on-policy counterpart of their deep Q-network replay. On PUTNEXTLOCAL the published configuration fails before the generator matters: the generator trains only on the agent's own successes, which number in the tens over the full budget.

HIGhER+BC. HIGhER+BC transplants the learned whole-task relabeler into our pipeline with the BC prior, a demonstration-seeded buffer, the same corpus, and the same budget. It differs from GLEAN in imitating its own episodes alongside the demonstrations and in what whole-task granularity requires: a flat prior, whole-instruction conditioning, and whole-episode segments. On the compositional levels, the buffer admits tens of thousands of episodes per run, mostly successful episodes imitated under their own instructions. The generator's held-out exact-match ends training at 0.09–0.26, so the trust gate is open for at most 30% of training on PUTNEXTLOCAL and never opens on GOTOSEQ. The HIGhER+BC results, including the GOTOSEQ lead, are therefore produced by imitating the agent's own successful episodes over the flat prior; generated relabels of failures play almost no part. An earlier configuration whose relabeler was always on, with no trust gate, reproduces every row within noise.

HIGhER+BC strengthens the published method; it is the strongest baseline in this study, the strongest non-oracle method on GOTOSEQ, and nominally ahead of GLEAN on MOVETWOACROSS. As published, HIGhER reaches about 0.33 success on a single-instruction fetch grid after 5M steps, against about 0.40 for hindsight experience replay with expert relabels.

A.3.4 ELLA

ELLA trains an offline FiLM termination classifier on (observation, subgoal) pairs from the same demonstration budget, with held-out accuracy 0.93 at the full corpus, alongside an online Siamese relevance classifier. It adds a per-completion bonus at the published values, rescaled to remove the platform's $\times 20$ reward scaling: 0.0125 ($0.25/20$, published for its put-next levels) on PUTNEXTLOCAL and PUTNEXTS5N2, and 0.025 ($0.5/20$, published for its longer-horizon open-and-pick and sequence levels) on GOTOSEQ and MOVETWOACROSS. ELLA evaluates PUTNEXTLOCAL and five custom two-room levels and uses GOTOLOCAL only as a low-level task set, so we assign GOTOLOCAL and OPENTWODOORS 0.0125 and 0.025 by horizon. The bonus keeps the original's per-timestep and per-instruction caps and its success-neutralization, and the policy sees the whole instruction.

Three things differ from the original. The shared RECURRENTPPO stack replaces the original FiLM–LSTM actor–critic, as for every baseline; the termination classifier trains on the shared demonstration corpus instead of dedicated low-level expert episodes; and apart from PUTNEXTLOCAL, which ELLA also evaluates, ELLA's levels map onto ours by analogy. Everything else follows the paper, including intersection-based deduplication of its store of low-level instructions, and the released code settles the sampling details the paper leaves open.

A.3.5 CAREL

CAREL trains online PPO on all trajectories and adds an X-CLIP-style symmetric InfoNCE auxiliary loss over four similarity granularities, computed on successful trajectories only (λ_C=0.01, attention-pooling temperature 1). It tracks the instruction with similarity-spike detection, which marks a subtask done when its score exceeds k=2 times its mean over earlier steps, and the published masking-probability ramp. It adds no auxiliary reward and uses no demonstrations. We make two adaptations: a trainable projection head over frozen instruction embeddings in place of a trained language encoder, and instruction tracking kept active at evaluation, following their algorithm's online loop (the released code evaluates on the unmasked whole instruction).

Table 5: Detector sensitivity at the full corpus: GLEAN + self-imitation + completion reward w/o filter-state input, with one detector change per row (three seeds; first row and reset filter five). "False-adv." = fraction of 200 held-out uniform-random rollouts per level (PUTNEXTLOCAL/GOTOSEQ) whose first detection precedes the true completion; "true-adv." = fraction of 200 BC-policy PUTNEXTLOCAL rollouts whose first detection is correct; "audit" = empirical false-advancement rate of the calibrated threshold on held-out nulls, worst (level, kind) cell, target α=0.05.

variant	GOTOLOCAL	PUTNEXTLOCAL	GOTOSEQ	offline measurement
unchanged detector	0.999±0.001	0.952±0.026	0.825±0.019	false-adv. 0.005/0.027; audit $\leq$ 0.058
absolute readout	0.995±0.002	0.740±0.266	0.802±0.016	false-adv. 0.020/0.013
reset filter	0.996±0.004	0.924±0.083	0.842±0.033	false-adv. 0.013/0.027
LSTM filter	0.999±0.001	0.938±0.054	0.793±0.004	false-adv. 0.015/0.015
no filter	0.996±0.002	0.960±0.005	0.770±0.027	single-frame evidence
no rollout negatives	0.998±0.003	0.965±0.013	0.809±0.023	false-adv. 0.065/0.098
thresholds pooled across levels	0.999±0.001	0.946±0.013	0.824±0.023	audit 0.13
1,500-episode detector	0.997±0.001	0.930±0.025	0.816±0.009	true-adv. 0.775 (first row 0.920)

A.4 DETECTOR DESIGN AND ERROR CONTROL

Figure 3 shows GLEAN's detector on a held-out episode: the score for "go to the box" crosses its calibrated threshold when the agent reaches the box, and the policy's instruction switches to "go to a ball".

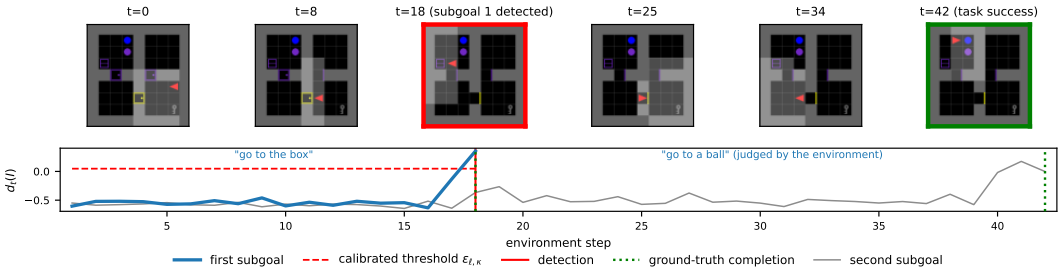

Figure 3: Completion detection on a held-out episode with the instruction "go to the box, then go to a ball". Top: full-grid renders for display; the detector and the policy see the egocentric view. The first subgoal is detected as complete at step 18 (red frame, red line), where the environment's completion predicate also becomes true (green dotted line); the second completes at step 42 (green frame, task success), which the environment judges. Bottom: the detector's score for the first subgoal while it is active (blue), its calibrated threshold (red dashed), and its score for the second subgoal (grey).

When completion detection was first restricted to the observation history, in an early configuration trained on a small corpus, PUTNEXTLOCAL collapsed in the online loop. The collapse came from false advancement on wandering trajectories, where 26% of uniform-random episodes detected "pick up" before any pickup; from a calibration window shorter than the online scoring window, where a target α=0.05 measured 0.14–0.21; and from a corpus one to two orders of magnitude below the level's published requirement. GLEAN's detector adds rollout negatives, calibrates per (level, kind) over the online scoring window, and trains on the full corpus. The online results in this section use GLEAN + self-imitation + completion reward w/o filter-state input, at three seeds unless stated. Table 5 ablates the rollout negatives, the per-cell calibration, and the corpus size; the calibration horizon is not ablated.

A.4.1 DETECTOR DESIGN

Table 5 changes one detector choice at a time. The absolute readout scores the filter state u_t itself in place of its change since the last completion; the reset filter restarts the filter at each advancement and keeps the difference readout; the LSTM filter replaces the causal transformer; no filter scores the difference between single-frame encodings at the reference and current steps; no rollout negatives drops the random rollouts from training; thresholds pooled across levels calibrate one threshold per subgoal kind; and the 1,500-episode detector is trained and calibrated on 1,500 demonstrations. The

unchanged row and the reset filter carry five seeds and the other variants three, so comparisons with a three-seed variant use the unchanged row's values over the same three seeds, 0.943 and 0.834.

Apart from the absolute readout, final success varies little across the detector variants: PUTNEXTLOCAL spans 0.924–0.965 and GOTOSEQ spans 0.770–0.842. On PUTNEXTLOCAL, only the absolute readout differs by more than two standard deviations (0.740 ± 0.266); on GOTOSEQ, removing the filter (0.770), replacing it with an LSTM (0.793), the absolute readout (0.802), and dropping the rollout negatives (0.809) cost more than two standard deviations of the unchanged detector over the same three seeds (0.834 ± 0.012). A 1,500-episode detector costs 0.013 and 0.018.

The detector variants differ mainly in offline error control. Training without rollout negatives raises false advancement on held-out random rollouts from 0.005 to 0.065 on PUTNEXTLOCAL and from 0.027 to 0.098 on GOTOSEQ, and thresholds pooled across levels lose per-cell error control, auditing at 0.13 in their worst cell against the $\alpha=0.05$ target. Neither changes PUTNEXTLOCAL final success.

A.4.2 READOUT SEMANTICS

Two ways of referencing the last completion improve on the absolute readout on PUTNEXTLOCAL: subtracting u_{ref}, GLEAN's difference readout, and resetting the filter at each advancement. Only the difference readout does so reliably. It reaches 0.952 ± 0.026, and the reset filter 0.924 ± 0.083, against 0.740 ± 0.266 for the absolute readout. On GOTOSEQ the reset filter matches the difference readout (0.842 ± 0.033 against 0.825 ± 0.019). This supports GLEAN's choice of a persistent filter with the reference state subtracted.

The offline probes of Table 6 separate the readouts the same way. True advancement on BC rollouts is 0.920 for the difference readout against 0.752 for the absolute readout. On held-out random rollouts neither readout is uniformly cleaner: the difference readout makes false detections less often on PUTNEXTLOCAL (0.005 against 0.020) and more often on GOTOSEQ (0.027 against 0.013). On the single-subgoal level the readouts coincide by construction, since no mid-episode advancement occurs.

A.4.3 OFFLINE PROBES OF THE SEQUENCE ESTIMATORS

Table 6 probes each detector variant offline, before any online training. Under the absolute readout, a persistent filter lowers true advancement for both sequence models. Moving from a filter reset at advancement to a persistent filter, the LSTM's true advancement falls from 0.915 to 0.670 and its *goto* recall from 0.40 to 0.29; the transformer's falls from 0.952 to 0.752 and from 0.43 to 0.33. The difference readout restores the persistent transformer to 0.920 and 0.39; combining the difference with a reset gives 0.958 and the highest *goto* recall of any variant, 0.46. With the difference readout, the LSTM filter probes at 0.962 and 0.41; its online row is in Table 5. Pooled over levels, every variant holds its held-out false-advancement rate per subgoal kind at 0.049–0.053 against $\alpha=0.05$; Table 5 reports the worst (level, kind) cell.

Adding action tokens to the detector's input lowers *goto* recall to 0.26 and true advancement to 0.707. Online, GLEAN + self-imitation + completion reward w/o filter-state input with action tokens ends at 0.994, 0.924, and 0.823 on GOTOLOCAL, PUTNEXTLOCAL, and GOTOSEQ (three, five, and three seeds). On OPENTWODOORS it ends at 0.655 ± 0.270 against 0.633 ± 0.185 without action tokens (five seeds each), within one standard deviation.

A.4.4 DETECTION COVERAGE VERSUS DETECTION ACCURACY

Coverage is the share of subgoals the detector credits, and accuracy is whether the credited completions are correct. On GOTOSEQ, GLEAN's detector has two coverage defects: it looks up subgoal texts by exact string where training parsed them, and its vocabulary has no door referents. Repairing the vocabulary raises the credited fraction on GOTOSEQ from 0.209 to 0.547 without raising final success (0.811 against 0.834), while oracle advancement, whose completions are correct by construction, raises final success by 0.079 in the same paired comparison (Table 7). Detection accuracy, not coverage, limits GOTOSEQ.

Table 6: Offline probes at the 20K corpus (three seeds, mean). Rows name their change from GLEAN's detector of Section 4. Recall = fraction of demonstrated completions detected on expert replay, per kind. False-adv. = fraction of 200 held-out uniform-random rollouts per level whose first detection precedes the true completion. True-adv. = fraction of 200 BC-policy PUTNEXTLOCAL rollouts whose first detection is correct.

	recall		false-adv.		true-adv.
variant	pickup	goto	PUTNEXTLOCAL	GOTOSEQ	PUTNEXTLOCAL
LSTM filter, absolute readout, reset filter	1.000	0.40	0.003	0.025	0.915
LSTM filter, absolute readout	1.000	0.29	0.007	0.013	0.670
absolute readout, reset filter	1.000	0.43	0.013	0.025	0.952
absolute readout	1.000	0.33	0.020	0.013	0.752
reset filter	1.000	0.46	0.013	0.027	0.958
LSTM filter	1.000	0.41	0.015	0.015	0.962
GLEAN's detector	0.998	0.39	0.005	0.027	0.920
action tokens	0.992	0.26	0.002	0.008	0.707

Table 7: Coverage vs accuracy (GLEAN + self-imitation + completion reward w/o filter-state input, three seeds; *credited* = fraction of each training episode's subgoals credited by the detector on GOTOSEQ).

detector	GOTOLOCAL	PUTNEXTLOCAL	GOTOSEQ	credited
GLEAN's detector	0.999	0.943	0.834	0.209
vocabulary repair	0.998	0.952	0.811	0.547
oracle advancement	0.999	0.966	**0.913**	—

A.5 ADDITIONAL EXPERIMENTS

A.5.1 LEARNING CURVES ON THREE LEVELS

Figure 4 adds MOVETWOACROSS, the four-subgoal level, to the learning curves of the main text. There GLEAN and HIGhER+BC also rise from their cloned start, every flat-prior variant that stops imitating collapses within the first 150K steps, and GLEAN w/o online imitation falls, as on GOTOSEQ.

A.5.2 FULL ABLATION TABLE

Table 8 extends Table 2 with GLEAN w/ self-imitation only, which replaces the demonstrations in online imitation with the agent's own detector-admitted segments, and with interaction variants: removing online imitation, adding self-imitation, and using self-imitation only, each with the completion reward added, and removing the filter-state input together with adding self-imitation or removing online imitation, both with the completion reward. With the completion reward and without online imitation, removing the filter-state input as well lowers final success from 0.906 to 0.742 on PUTNEXTLOCAL and from 0.457 to 0.348 on GOTOSEQ and makes PUTNEXTLOCAL unstable across seeds (± 0.188). With self-imitation and the completion reward, the filter-state input raises final success on OPENTWODOORS, PUTNEXTS5N2, and MOVETWOACROSS by 0.242, 0.027, and 0.100. Self-imitation only stays within 0.05 of GLEAN on GOTOLOCAL, PUTNEXTLOCAL, and GOTOSEQ, but underperforms GLEAN w/o online imitation on OPENTWODOORS and PUTNEXTS5N2 (0.445 against 0.870 and 0.800 against 0.993).

A.5.3 ONLINE-HYPERPARAMETER SENSITIVITY

In Table 9, the α variants re-derive thresholds from stored calibration material and leave the detector's weights unchanged. Its first row reports the same five seeds as the variants, so every comparison in it is paired.

On PUTNEXTLOCAL the stricter advancement target (α=.01) costs 0.037 and widens the seed spread (0.936 ± 0.046), and 16 online-imitation batches cost 0.025; every other setting stays within 0.010 of the first row. On GOTOSEQ, 16 online-imitation batches cost 0.048 and 256 cost 0.030, both advancement targets stay within 0.006 of the first row, and 32 to 128 batches stay within 0.012.

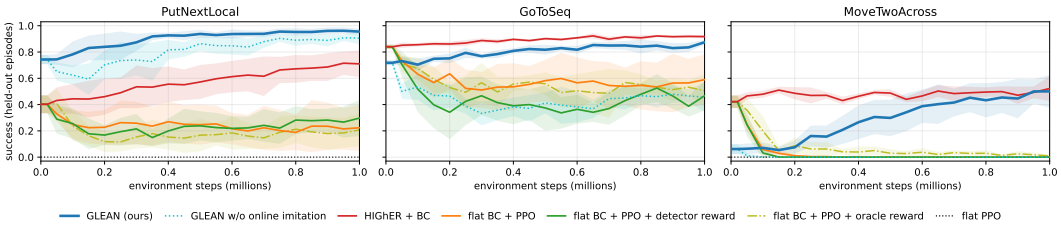

Figure 4: Learning curves on three levels (mean±s.d. over five seeds; evaluated every 50K steps on 256 held-out episodes, final point on 512). Step 0 is the behavior-cloned initialization, shared by GLEAN and GLEAN w/o online imitation and by the four flat-prior variants; flat PPO starts from a random policy.

Table 8: All ablations at the full corpus, including the interaction variants: final success, mean±s.d. over five seeds. Each row below GLEAN names its change to GLEAN. Top: component removals and the completion reward. Middle: imitation data, with and without the completion reward. Bottom: policy input, alone and combined with other changes.

variant	GoToLocal	PutNextLocal	GoToSeq	OpenTwoDoors	PutNextS5N2	MoveTwoAcross
GLEAN	.996±.001	.973±.011	.891±.014	.996±.005	.995±.006	.469±.147
w/o BC init	.993±.003	.322±.173	.848±.021	.889±.205	.773±.201	.021±.017
w/o online imitation	.990±.004	.892±.023	.336±.246	.870±.247	.993±.004	.000±.000
w/o critic warm-up	.995±.002	.973±.003	.880±.029	.999±.002	.993±.003	.424±.139
+ completion reward	.995±.002	.955±.022	.874±.018	.991±.011	.985±.004	.500±.125
w/o advancement or filter-state input, + completion reward	.996±.001	.836±.015	.894±.013	.687±.236	.955±.021	.784±.098
+ self-imitation	.996±.003	.968±.016	.889±.021	.911±.105	.995±.003	.105±.046
+ self-imitation + completion reward	.996±.002	.974±.008	.858±.024	.875±.163	.970±.017	.240±.152
w/ self-imitation only	.996±.002	.947±.027	.844±.014	.445±.168	.800±.235	.025±.020
w/ self-imitation only, + completion reward	.996±.002	.951±.020	.849±.019	.563±.255	.725±.284	.073±.057
w/o online imitation, + completion reward	.990±.004	.906±.047	.457±.111	.979±.030	.906±.078	.000±.000
w/o filter-state input	.995±.004	.927±.062	.817±.036	.998±.004	.992±.012	.248±.129
+ self-imitation + completion reward w/o filter-state input	.999±.001	.952±.026	.825±.019	.633±.185	.943±.033	.140±.083
w/o filter-state input or online imitation, + completion reward	.995±.004	.742±.188	.348±.144	—	—	—
w/o observation input	.651±.038	.007±.002	.350±.051	1.000±.000	.048±.020	.017±.020
w/ filter-change input	.995±.001	.968±.005	.871±.018	1.000±.000	.996±.001	.101±.079

A.5.4 The demonstration-budget sweep, tabulated

Table 10 tabulates the demonstration-budget sweep for GLEAN and three baselines at six budgets, and Figure 5 plots it on all three levels.

On PutNextLocal, every method stays below 0.05 at 100 episodes, and the margin of GLEAN is largest mid-sweep: at 5,000 episodes it reaches 0.856, above every baseline's full-corpus result. On BC-policy rollouts, before reinforcement learning starts, the 10,000-episode detectors reach true advancement 0.945 and the 5,000-episode detectors 0.735, both with no false advancement.

Self-imitation does not help at reduced budgets either; both comparisons below provide the completion reward. With the filter-state input (five seeds), adding self-imitation changes PutNextLocal final success at 500, 1,500, and 5,000 demonstrations by $+0.059$, -0.059, and -0.156, so it helps only at 500 demonstrations. Without the filter-state input (three seeds), it changes final success by $+0.010$, -0.077, and $+0.024$, every difference inside one standard deviation.

Every corpus reported here realizes its target episode count; an earlier GoToSeq corpus that fell short of the target (9,794 of 10,000 episodes) gave the same ordering.

A.5.5 Pure-imitation scaling and one-axis corpus-size sweeps

Table 11 reports pure behavior cloning at the sweep's budgets under flat and subgoal conditioning, and Table 12 the one-axis corpus-size sweeps that reduce the policy-side data or the detector corpus alone. With oracle advancement, subgoal BC scales well past flat BC on the compositional levels; with detector advancement, the GoToSeq priors level off near 0.71 from 5,000 episodes, so *goto* detection limits the cloned prior on that level before any online training.

Table 9: Online-hyperparameter sensitivity of GLEAN (five seeds; first row = GLEAN at α=.05 with 64 online-imitation batches).

setting	PUTNEXTLOCAL	GOTOSEQ
unchanged settings	.973±.011	.891±.014
advancement α=.01	.936±.046	.885±.022
advancement α=.10	.974±.007	.885±.024
16 online-imitation batches	.948±.004	.843±.020
32 online-imitation batches	.963±.011	.897±.009
128 online-imitation batches	.970±.012	.879±.031
256 online-imitation batches	.971±.024	.861±.039

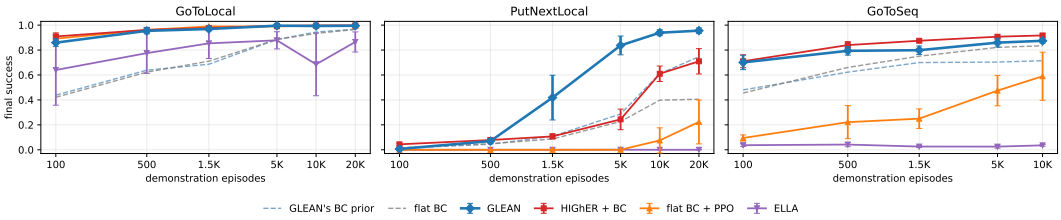

Figure 5: Demonstration-budget sweep (mean±s.d. over five seeds). Dashed: flat BC and GLEAN's BC prior before online training.

Rebuilt without self-imitation, so that the online-imitation buffer holds demonstrations only, the policy-data cells of Table 12 move within one standard deviation in both directions ($0.232 \rightarrow 0.186$ at 500 demonstrations and $0.516 \rightarrow 0.588$ at 1,500).

A.5.6 RETENTION WITHOUT ONLINE IMITATION

We swap the retention mechanism on PUTNEXTLOCAL at the full corpus, under the earlier n=200 protocol, in an earlier configuration with the difference readout and the completion reward and without the filter-state input (three seeds per mechanism). Kickstarting-style distillation from the frozen BC teacher reaches 0.915 ± 0.029 at a constant weight and 0.932 ± 0.014 annealed, against 0.940 ± 0.022 for online imitation with self-imitation, and with neither mechanism the policy reaches 0.913 ± 0.021. The mechanisms are interchangeable at the mean, and at this corpus size the difference readout keeps the cloned prior without either. With the absolute readout and neither mechanism, the policy ends at 0.870 ± 0.124, and either mechanism removes that loss.

The subgoal components matter more: flat BC+PPO reaches 0.200 ± 0.147 in this protocol, below its imitation-only success. Distillation keeps the frozen teacher in the loss for the whole run, whereas online imitation keeps imitating the demonstrations.

A.6 FORMAL ANALYSIS

A.6.1 LEMMA 1: OPTIMIZATION OF NECESSARY SUBGOALS

Our analysis begins with the fact that under the instructions given by BabyAI, all subgoals are strictly necessary for the completion of the episode. Let s_{I_i} represent the state that marks the completion of instruction I_i, for $i \in \{1, \ldots, K\}$. Under this representation, the condition for successfully terminating an episode is $s_t = s_{I_K}$. Given that BabyAI's instruction decompositions represent necessary steps to completing the episode, we can say that:

$$s_t = s_{I_K} \implies \forall i \in \{1, \ldots K-1\}, \exists t_i : s_{t_i} = s_{I_i}$$

If the goals are ordered, then we may also impose the restriction that $t_1 \leq t_2 \leq \cdots \leq t_{K-1}$. In the case of BabyAI, almost all tasks are strictly ordered. For those with ambiguous ordering, the optimality of our method is predicated on the optimality of the order within the instruction decomposition.

Table 10: Demonstration-budget sweep (five seeds per cell; GoToSeq's full corpus is 10,000 episodes). ELLA is its published configuration with only the termination classifier retrained per budget. ELLA's termination classifier pools *goto* segments from all three levels, so its GoToSeq 10,000 cell (every level at 10,000 episodes) differs from its full-corpus run (the other two levels at 20,000, as in Table 1).

level	method	100	500	1,500	5,000	10,000	full
GoToLocal	GLEAN	.859±.030	.954±.024	.969±.014	**.995±.002**	.993±.004	.996±.001
	HIGhER+BC	**.909±.029**	**.962±.015**	.977±.013	.994±.003	**.997±.002**	**.998±.002**
	flat BC+PPO	.891±.034	.961±.019	**.989±.007**	.990±.003	.995±.001	.994±.004
	ELLA	.640±.282	.775±.163	.854±.121	.878±.069	.686±.252	.865±.081
PutNextLocal	GLEAN	.010±.005	.054±.023	**.252±.160**	**.856±.063**	**.942±.025**	**.973±.011**
	HIGhER+BC	**.044±.009**	**.078±.019**	.107±.020	.244±.083	.610±.062	.710±.102
	flat BC+PPO	.000±.000	.000±.000	.000±.000	.000±.001	.075±.101	.223±.176
	ELLA	.000±.000	.000±.000	.000±.000	.000±.001	.000±.000	.000±.000
GoToSeq	GLEAN	.682±.122	.809±.025	.836±.024	.875±.021	.891±.014	—
	HIGhER+BC	**.710±.052**	**.840±.028**	**.875±.011**	**.907±.013**	**.917±.012**	—
	flat BC+PPO	.095±.025	.221±.133	.250±.078	.475±.122	.590±.193	—
	ELLA	.037±.011	.042±.017	.026±.008	.025±.013	.036±.007	.023±.010

Table 11: Pure-BC demonstration-budget scaling, final success (five seeds; GoToSeq full = its 10,000-episode corpus). GLEAN's BC prior is subgoal BC with the filter-state input, advanced by the detector. On GoToLocal, detector advancement coincides with oracle advancement, since the environment judges the only subgoal.

level	variant	100	500	1,500	5,000	10,000	full
GoToLocal	flat BC	.421	.623	.711	.885	.933	.965
	subgoal BC w/ oracle advancement	.422	.636	.721	.889	.932	.971
	GLEAN's BC prior	.438	.638	.686	.885	.945	.967
PutNextLocal	flat BC	.016	.048	.087	.227	.398	.405
	subgoal BC w/ oracle advancement	.018	.062	.135	.441	.615	.752
	subgoal BC w/ detector advancement	.005	.050	.116	.362	.596	.739
	GLEAN's BC prior	.011	.045	.108	.287	.611	.746
GoToSeq	flat BC	.455	.661	.751	.821	.834	—
	subgoal BC w/ oracle advancement	.481	.746	.830	.921	.956	—
	subgoal BC w/ detector advancement	.450	.627	.677	.718	.713	—
	GLEAN's BC prior	.480	.623	.700	.704	.714	—

Therefore, under strict ordering we may separate any successful state trajectory τ into sub-trajectories $\tau_i = \{s_{I_{i-1}}, \ldots, s_{I_i}\}$, where by convention, $s_{I_0} = s_0$. By conditional independence of MDPs, we can be certain that the dynamics within each τ_i are dependent only on their initial state $s_{I_{i-1}}$, and that there is no need to revisit that state within the same sub-trajectory, hence we only provide the reward for completing a subgoal once. This conditional independence also means that we may optimize each subgoal independently from others. Since the only reward for each sub-trajectory is given for completing it, optimizing the reward of each sub-trajectory means minimizing the time to complete it, and since the sub-trajectories are independent, minimizing the time of the sub-trajectories is the same as minimizing the overall time for completing the episode, and therefore maximizing the discounted reward.

Therefore, given the instructions accurately identify necessary states for completing the episode (or identify states that must be achieved during optimal execution), and that these states are decomposed into an optimal order of subgoals, optimizing to achieve these subgoals in minimal time is synonymous with optimizing the discounted reward of the episode.

A.6.2 Lemma 2: Subgoal Embedding

Since completion of these subgoals is not indicated by the environment, we must train g_η so that $\psi(s_0, s_{t_i}) = \sum_{j=1}^{i} z_j = \sum_{j=1}^{i} g_\eta(\text{enc}(I_j))$. The contrastive InfoNCE loss (van den Oord et al., 2018) specifically aligns the embeddings g_η by minimizing $\sum_j \mathcal{L}_{\text{ground}}(g_\eta(\text{enc}(I_j)), \psi(s_{I_{j-1}}, s_{I_j}))$. InfoNCE incorporates two terms of entropy of softmaxes, so it is a convex loss function. By Jensen's inequality:

Table 12: One-axis corpus-size sweeps (three seeds, GLEAN + self-imitation + completion reward w/o filter-state input; the other axis stays at the full corpus).

level	reduced axis	100	500	1,500	5,000
PUTNEXTLOCAL	policy data	.036±.007	.232±.049	.516±.172	.780±.099
	detector corpus	.730±.074	.765±.134	.930±.025	.872±.120
GOTOSEQ	policy data	.592±.040	.727±.019	.725±.021	.785±.043
	detector corpus	.834±.006	.820±.015	.816±.009	.839±.014

$$\mathcal{L}_{\text{ground}}(g_\eta(\text{enc}(I_i)), \psi(s_0, s_{I_i}))$$
$$\leq \sum_{j=1}^{i} \mathcal{L}_{\text{ground}}(g_\eta(\text{enc}(I_j)), \psi(s_{I_{j-1}}, s_{I_j})) \tag{8}$$

Therefore, we can be certain that the loss for representing the trajectory $\{s_0, s_{I_i}\}$ is no greater than the sum of losses for representing all instructions preceding I_i, and that we may use a sufficiently trained g_η to evaluate instructions as being completed based on differences in state embeddings, as long as the instructions are correctly matched to the states within the offline data used to train g_η.

A.6.3 LEMMA 3: CONFORMAL GATE

So far we have proven that exact matches within the embedding space are sufficient to prove that the subgoal has been completed. Since g_η is evaluated as having minimized but almost certainly nonzero entropy, it is a continuous probability distribution, meaning that exact matches are almost certainly impossible. Instead, we identify an error margin expressed as a minimum cosine similarity that the effect embedding ψ must achieve for the subgoal to be evaluated as complete. The split conformal approach (Papadopoulos et al., 2002; Lei et al., 2018) outlined in equation 3 of the main paper uses a held-out set of observations drawn from the same distribution to approximate a confidence interval of $(1 - \alpha)$, and as such, we can be certain that an embedding that falls within the conformal gate has achieved the subgoal with confidence $(1 - \alpha)$, albeit with the possibility of compounding error unless the previous error is always subtracted from the next subgoal embedding, a consideration that was not yet necessary for our results.

A.6.4 THEOREM 1: CONFORMAL GATES ON INSTRUCTION EMBEDDINGS

Consider a BabyAI task with instructions which accurately identify states which are either necessary for episode completion or are optimal for episode completion. Given that the instructions decompose into an optimal order, then by Lemma 1, we can be certain that minimizing the time between achieving each subgoal is synonymous with maximizing discounted reward in the episode. Then, as long as sub-trajectory embeddings fall within the conformal gate, we can be $(1 - \alpha)$ certain that a subgoal has been achieved (Lemma 3), and that we may proceed to the next subgoal, where the overall goal is a sum of subgoals (Lemma 2), and where we need to only provide the reward for completing a subgoal once (Lemma 1). Under these conditions, optimizing for the rewards returned once for each completed subgoal is synonymous with optimizing for the overall episode with a worst case confidence of $(1 - \alpha)^K$.

A.6.5 HINDSIGHT RELABELING ALLOWS FOR ONLINE EXPLORATION OF SUBGOAL OPTIMIZATION

While the offline trajectories do accurately represent subskills for subgoal execution, a policy trained purely from the offline trajectories would be subject to value overestimation as a result of offline-online transfer (Kumar et al., 2020; Nair et al., 2020), meaning that we must use the online stage to gather trajectories explored with an online policy and label them as corresponding to the subgoals using hindsight relabeling (Sec. 4.3 of the main paper). Since the trajectories are gathered by a policy trained to multitask between on-policy exploration with PPO and off-policy behaviour cloning, the new gathered trajectories are subject to less action distribution shift than the fully offline expert demonstrations, and allows us to mitigate the action shift from deploying from offline to online. In

the case of a PPO policy that has converged on an optimum behaviour, a sufficient density of its relabeled trajectories from successful episodes will eventually cause the behaviour cloning term to begin cloning the PPO policy. In this sense, the behavioural cloning term can be considered a moving average of the PPO policies, similar to what is standard for off-policy reinforcement learning. As implemented, this holds up to the protected demonstration fraction λ of Eq. (8) of the main paper: the demonstration tier is never evicted, so the cloning term converges to a mixture of the expert corpus and the agent's own relabeled successes rather than onto the PPO policy alone.