# ReWiND: Language-Guided Rewards Teach Robot Policies without New Demonstrations

Jiahui Zhang*[1], Yusen Luo*[1], Abrar Anwar*[1],
Sumedh A. Sontakke[2], Joseph J. Lim[3], Jesse Thomason[1], Erdem Bıyık[1], Jesse Zhang[1]

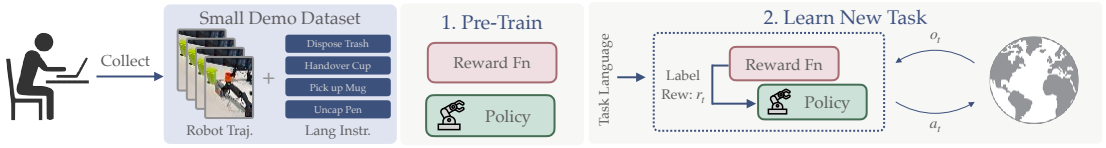



Fig. 1: We pre-train a policy and reward model from few language-labeled demos to solve unseen task variations via lang-guided RL *without new demos*.

***Abstract*—We introduce ReWiND, a framework for learning robot manipulation tasks solely from language instructions *without per-task demonstrations*. Standard reinforcement learning (RL) and imitation learning methods require expert supervision through human-designed reward functions or demonstrations for every new task. In contrast, ReWiND starts from a small demonstration dataset to learn: (1) a data-efficient, language-conditioned reward function that labels the dataset with rewards, and (2) a language-conditioned policy pre-trained with offline RL using these rewards. Given an unseen task variation, ReWiND fine-tunes the pre-trained policy using the learned reward function, requiring minimal online interaction. We show that ReWiND's reward model generalizes effectively to unseen tasks, outperforming baselines by up to $2.4\times$ in reward generalization and policy alignment metrics. Finally, we demonstrate that ReWiND enables sample-efficient adaptation to new tasks, beating baselines by $2\times$ in simulation and improving real-world pretrained bimanual policies by $5\times$, taking a step towards scalable, real-world robot learning. See website at https://rewind-reward.github.io/.**

## I. Introduction

A great teacher does not just tell you if you are right or wrong. Instead, they guide you by providing feedback when you make mistakes, highlighting progress as you learn something new, and adapting to how you learn best. For deployed robots to learn new tasks in the wild, they need similarly intelligent teachers. These teachers—in the form of robust reward models—should: (1) offer *dense, informative feedback*, especially during failures; (2) *generalize* their guidance to unseen tasks; and (3) remain *robust* to diverse robot behaviors during its learning process. Our paper leverages these insights to develop reward models that can teach robots unseen tasks.

In this work, we introduce ReWiND (**Re**wards **Wi**thout **N**ew **D**emonstrations), a framework designed to teach robots unseen tasks in a sample-efficient manner using only a few grounding human demonstrations for training tasks (see Figure 1). Typically, teaching robots involves large-scale imitation learning [1]–[4], where human experts provide demonstrations for each new task. However, collecting task-specific demonstrations is expensive and time-consuming. Reinforcement learning (RL) offers a more autonomous alternative by using reward functions as teachers, allowing robots to learn through interaction. Yet, manually designing these reward functions demands substantial manual effort and domain-specific expertise [5]. Recent progress in language-conditioned reward learning [6]–[15] has aimed at addressing these challenges, but often assumes unrealistic conditions such as availability of ground-truth states [7]–[12], thousands of demonstrations [13], or online training of reward models from scratch [14], [15], limiting their practical applicability.

ReWiND overcomes these challenges by instead assuming only a handful of demonstrations—e.g., five per task—to enable real-world robot learning of unseen task variations. ReWiND first trains a language-conditioned reward model from these demonstrations, then uses it to pre-train a language-conditioned policy via offline RL. When deployed, ReWiND efficiently fine-tunes the policy on new task variations by reward-labeling *online* interaction episodes.

Our core contribution is in designing ReWiND's reward model to capture three key properties outlined earlier: **dense feedback**, **generalization**, and **robustness**. First, to provide *dense, informative feedback*, we design a *cross-modal sequential aggregator* that predicts *progress* within demonstration videos. Progress prediction offers a densely supervised training signal that naturally translates into rewards. We also introduce *video rewinding* to automatically generate failure trajectories from successful demos, allowing ReWiND to provide dense feedback even when the policy is making mistakes. Then, to encourage *generalization* across unseen tasks and *robustness* to diverse behaviors, we train the cross-modal sequential aggregator with pre-trained vision and language embeddings, selectively applied positional embeddings, and diverse robotics data from Open-X dataset [16] . Focusing on these properties enables ReWiND rewards to extrapolate to novel visual and language inputs.

We introduce reward metrics measuring the above properties on which ReWiND achieves **23-74%** relative improvements over reward learning baselines. Further, comprehensive success rate evaluations on Metaworld manipulation tasks and a real-world bimanual robot setup demonstrate ReWiND beats baselines by **2x** in simulation and improves real-world pre-trained policies by **5x**.

[1]University of Southern California, [2]Amazon Robotics, [3]KAIST

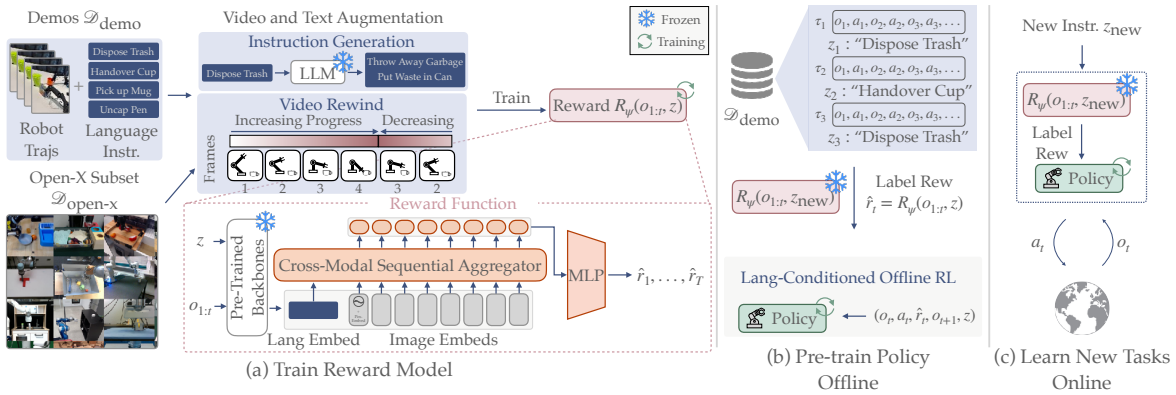



Fig. 2: **(a):** We train a reward model $R_\psi(o_{1:t}, z)$ on a small demonstration dataset $\mathcal{D}_{\text{demos}}$ and a curated subset of Open-X, $\mathcal{D}_{\text{open-x}}$, augmented with LLM-generated instructions and video rewinding. $R_\psi(o_{1:t}, z)$ predicts video *progress* rewards $\hat{r}_{1:T}$ from pre-trained embeddings of image observations $o_{1:T}$ and language instructions $z$, and assigns 0 progress to misaligned video-language pairs. **(b):** We use the trained $R_\psi(o_{1:t}, z)$ to label $\mathcal{D}_{\text{demos}}$ with rewards and pre-train a language-conditioned policy using offline RL. **(c):** For an unseen task specified by $z_{\text{new}}$, we fine-tune $\pi$ with online rollouts and reward labels from $R_\psi(o_{1:t}, z_{\text{new}})$.

## II. ReWiND: Learning Rewards Without New Demonstrations

We study the problem of learning unseen, language-specified tasks in a *target environment*, formulated as a Markov decision process (MDP). The *target environment* refers to the deployment scene (e.g., a robot tabletop). We train a policy $\pi_\theta(a_t \mid o_t, z)$ that selects actions $a_t$ based on images $o_t$ and language instructions $z$.The policy is optimized to maximize rewards predicted by a learned reward function $R_\psi(o_{1:t}, z)$, which conditions on the frame sequence $o_{1:t}$ and instruction $z$ to output per-timestep estimated rewards $\hat{r}_t$. We assume access to a small demonstration dataset $\mathcal{D}_{\text{demos}}$ in the target environment containing 15–20 tasks with $\sim$5 demonstrations each. Following prior definitions of generalization [17], [18], we define a task as unseen if it requires a novel *action sequence*, its distribution of *image observations* has changed, or needs a new *language instruction*.

ReWiND consists of 3 phases (see Figure 2): (1) learning a reward function from limited target environment demos, then (2) pre-training $\pi$ with learned rewards on the demos, and finally (3) using the reward function and pre-trained policy to learn a new language-specified task online.

### A. Learning a Reward Function

Our primary objective for reward prediction is regressing directly to per-frame *progress* within an observation sequence $o_{1:T}$ conditioned on instruction $z$. Unlike prior methods using relative targets [13], [19], our progress-based objective provides fixed targets that are more stable to train on, and translates directly into a dense, $[0, 1]$-normalized reward for policy training. To ensure robustness against mismatched observations and instructions, we also sample unrelated observation sequences $o^{\text{other}}_{1:T}$ and train $R_\psi(o_{1:t}, z)$ to predict zero progress. Our reward prediction loss is:

$$\mathcal{L}_{\text{progress}}(o_{1:T}, z, o^{\text{other}}_{1:T}) = \sum_{t=1}^{T} \left( R_\psi(o_{1:t}, z) - \underbrace{t/T}_{\text{matched seq. progress}} \right)^2 + \sum_{t=1}^{T} \underbrace{R_\psi(o^{\text{other}}_{1:t}, z)^2}_{\text{mismatched seq. 0 progress}} . \quad (1)$$

However, simply training a neural network $R_\psi(o_{1:t}, z)$ on $\mathcal{L}_{\text{progress}}(o_{1:T}, z, o^{\text{other}}_{1:T})$ with a small set of demonstrations is unlikely to ensure that it can train a policy on unseen tasks. $R_\psi(o_{1:t}, z)$ should:

- **D1 Generalize to new tasks**, i.e., new policy execution videos and instructions not in $\mathcal{D}_{\text{demos}}$.
- **D2 Produce rewards aligned with *policy rollouts***, not just successful demonstration videos.
- **D3 Be robust to input variations**, i.e., different ways to solve or specify the task.

To this end, we introduce a set of design choices spanning the training dataset, model architecture, and video and language augmentations that address all three desiderata. Specifically, we curate diverse off-the-shelf data from the Open-X dataset [16] to promote generalization (D1) and robustness (D3); apply targeted video and language augmentations for better reward prediction and language input robustness (D2, D3); and adopt specific network architectural modifications aimed at improving generalization (D1). For a visual overview, see Figure 2a.

**Incorporating Diverse Data (D1, D3).** To help $R_\psi(o_{1:t}, z)$ generalize to tasks unseen in $\mathcal{D}_{\text{demos}}$ (D1) and make it robust to diverse ways of executing and specifying tasks (D3), we subsample the Open-X Dataset [16], denoted $\mathcal{D}_{\text{open-x}}$. We specifically select Open-X trajectories with object-centric language instructions, e.g., "*pick coke can from fridge*," or directional instructions, e.g., "*drag the circle to the left of the star*," to help $R_\psi(o_{1:t}, z)$ generalize to objects and directions not contained in $\mathcal{D}_{\text{demos}}$. This dataset contains $\sim$356k

trajectories with ∼59k unique task strings. For detailed dataset information, see Section A-A1.

*1) Video and Language Augmentation (**D2**, **D3**):* Given our datasets $\mathcal{D}_{\text{demos}}$ and $\mathcal{D}_{\text{open-x}}$, we perform both video and language augmentations that help the reward function accurately predict rewards for unsuccessful *policy execution* videos (**D2**) and be robust to varied ways of specifying the task instructions $z$ (**D3**). We call the video augmentation **video rewind** and our text augmentation **instruction generation**.

**Video Rewind.** Both $\mathcal{D}_{\text{demos}}$ and $\mathcal{D}_{\text{open-x}}$ contain human demonstrations, which are assumed to be successful and of high-quality. Training $R_\psi(o_{1:t}, z)$ on $\mathcal{L}_{\text{progress}}(o_{1:T}, z, o^{\text{other}}_{1:T})$ only using these successful demonstrations, may result in $R_\psi(o_{1:t}, z)$ overfitting to these successful trajectories. However, during online deployment, $R_\psi(o_{1:t}, z)$ will likely encounter failure trajectories (unseen during training) which such an overfit model may reward highly. This is undesirable and prior works attempt to address this issue by explicitly training their reward model on failed trajectories [13], but these trajectories add a great additional burden on demonstrators to collect and must be added post-hoc to any existing dataset, making it harder to scale.

Instead, we address this problem in a scalable manner by randomly *rewinding videos*. Consider a video of a robot picking up a cup. If we rewind the video for a few frames right when the robot grabs the cup, it now looks like one in which the robot attempted to grasp the cup and then dropped it.[1] By training $R_\psi(o_{1:t}, z)$ to predict rewards corresponding to *reverse progress* on the rewound subsequence, it (1) is trained on observation sequences mimicking failed policy rollouts that will occur during online RL, and (2) learns to *decrease* reward when necessary. Thus rewinding helps $R_\psi(o_{1:t}, z)$ reward a *policy's* failures which will help with online RL (**D2**). See Figure 3 for an example. Formally, rewinding means sampling a random split point $i$ within an observation sequence $o_1 \ldots o_T$, rewinding $k$ ($k$ is also sampled) frames, then concatenating those $k$ frames to the end of the original sequence to become $o_1 \ldots o_i, o_{i-1}, \ldots, o_{i-k}$. The remaining frames from $i+1$ to $T$ are then unused. Our video rewind training objective follows:

$$\mathcal{L}_{\text{rewind}}(o_{1:T}, z) = \sum_{t=1}^{i} \underbrace{\left(R_\psi(o_{1:t}, z) - \frac{t}{T}\right)^2}_{\text{Loss for original trajectory until } i} + \sum_{t=1}^{k} \underbrace{\left(R_\psi([o_{1:i}, o_{i-1:i-t}], z) - \frac{i-t}{T}\right)^2}_{\text{Rewound video for } k \text{ frames from } i-1}. \quad (2)$$

**Instruction Generation.** We also generate 5-10 additional language instructions for each task in $\mathcal{D}_{\text{demos}}$ by prompting an LLM. This augmentation helps $R_\psi(o_{1:t}, z)$ with input robustness to possible new task instructions (**D3**). While training $R_\psi(o_{1:t}, z)$, any time we sample an observation sequence $o_{1:T}$, its instruction $z$ is uniformly randomly sampled from all available matching instructions, generated or original. We did not augment $\mathcal{D}_{\text{open-x}}$ due to its instruction diversity.

[1]Random rewinding may result in some physically implausible sequences. However, since they won't appear during inference, the rewards produced by $R_\psi(o_{1:t}, z)$ for such sequences should not affect online RL.

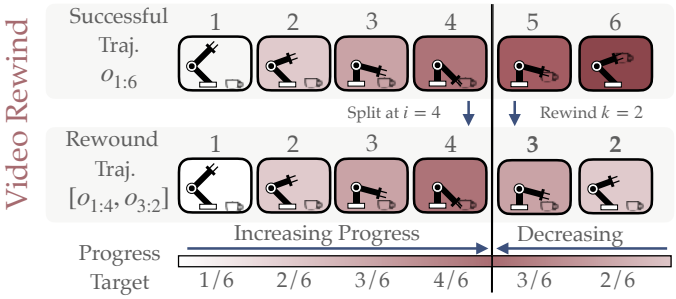



Fig. 3: **Video rewind.** We split a demo at intermediate timestep $i$ into forward/reverse sections. Here, the forward section shows the robot approaching the cup; the reverse section $(o_{i-1}, o_{i-2}, \ldots)$ resembles dropping it.

*2) Architecture (**D1**):* Due to the limited size of $\mathcal{D}_{\text{demos}}$, we carefully design the architecture for $R_\psi(o_{1:t}, z)$ to maximize generalization to new tasks (**D1**) while retaining the ability to optimize $\mathcal{L}_{\text{progress}}(o_{1:T}, z, o^{\text{other}}_{1:T})$ well.

**Frozen Input Encoders.** We use frozen image and language encoders as the backbone of $R_\psi(o_{1:t}, z)$: we use DINOv2 [20] for image encoding due to its strong object-centric representations and `all-MiniLM-L12-v2` [21] for instruction encoding due to its small embedding size ($= 384$). In $R_\psi(o_{1:t}, z)$, we first encode images and instructions: $o^{\text{embed}}_{1:t} = \text{DINO}(o_{1:t}), z^{\text{embed}} = \text{MiniLM}(z)$. Then, we train a small *cross-modal sequential aggregator* transformer conditioned on $(o^{\text{embed}}_{1:t}, z^{\text{embed}})$ that learns to aggregate frozen language and image embeddings to generate progress rewards $\hat{r}_t$ directly (see Figure 2(a) in the "Reward Function" box).

**Positional Embeddings.** Finally, the cross-modal sequential aggregator's transformer requires positional information about the frames to properly predict rewards (e.g., for distinguishing "pull" vs. "push"). However, if we naïvely add positional embeddings to each image, it can "cheat" by predicting progress using the positional embeddings. Therefore, similar to how [22] prompt an LLM with the position of the first video frame, we add a positional embedding to the first image.

**Reward Model Summary.** In summary, ReWiND trains a reward function $R_\psi(o_{1:t}, z)$ to predict task progress, using data augmentation (video rewinding and instruction generation) and additional Open-X data ($\mathcal{D}_{\text{open-x}}$) to improve generalization. For full implementation details, see Section A-A2. The final objective is:

$$\min_\psi \; \mathbb{E}_{(o_{1:T}, z, o^{\text{other}}_{1:T})} \left[\mathcal{L}_{\text{progress}}(o_{1:T}, z, o^{\text{other}}_{1:T}) + \mathcal{L}_{\text{rewind}}(o_{1:T}, z)\right]. \quad (3)$$

## B. Policy Learning

**Pre-training.** After training $R_\psi(o_{1:t}, z)$, we pre-train $\pi_\theta(a_t \mid o_t, z)$ on demonstrations $\mathcal{D}_{\text{demos}}$ labeled with rewards. This pre-training guides $\pi_\theta(a_t \mid o_t, z)$ toward reasonable behaviors during exploration, even if downstream tasks differ from those in $\mathcal{D}_{\text{demos}}$. Given a trajectory with instruction $z$, $\{(o_t, a_t)\}_1^T$, we assign rewards $\hat{r}_t = R_\psi(o_{1:t}, z)$ at each timestep and add success bonus to the final reward to encourage reaching the goal despite possibly noisy reward signals:

$$\hat{r}^{\text{off}}_t = R_\psi(o_{1:t}, z) + r_{\text{success}} \cdot \mathbb{1}[t = T]. \quad (4)$$

We then train $\pi_\theta(a_t \mid o_t, z)$ via offline RL using tuples $(o_t, a_t, \hat{r}_t, o_{t+1}, z)$. We use IQL [23] as prior work has demonstrated it works on real robots [24]–[26]. See Figure 2(b) for an overview.

**Learning Online.** To learn a new task online, ReWiND only requires a language description of the task, $z_{\text{new}}$. ReWiND rolls out $\pi(a \mid o_t, z_{\text{new}})$ and fine-tunes it on rewards coming from $R_\psi(o_{1:t}, z_{\text{new}})$. Like prior work [13], [19], we assume access to a success signal during online RL. We use this signal to give $r_{\text{success}}$ bonuses similar to in pre-training.[2] Our online rewards $\hat{r}^{\text{on}}$ are:

$$\hat{r}_t^{\text{on}} = R_\psi(o_{1:t}, z) + r_{\text{success}} \cdot \mathbb{1}[\text{success at } t]. \tag{5}$$

See full implementation details in Section A-A and pseudocode in Algorithm 1.

## III. Experiments

Our experiments aim to study the efficacy of ReWiND as a reward learning pipeline, evaluate its ability to train robots to learn new tasks efficiently, and analyze its design choices and limitations. To this end, we organize our experiments to answer the following empirical questions, in order:

- **(Q1) Rewards**: How well do ReWiND rewards correlate with task progress and success?
- **(Q2) Policy Learning**: Can ReWiND quickly train policies for new tasks?
- **(Q3) Ablations and Analysis**: Which ReWiND design decisions are most significant?

### A. Q1: What Makes a Good Reward Function?

We repeat the desiderata from Section II-A that we set out to achieve with ReWiND: (1) generalization to new tasks, (2) rewards aligned with videos from *policy rollouts*, and (3) robustness to diverse inputs. We structure this section to demonstrate ReWiND's ability to satisfy these criteria.

We compare ReWiND-learned rewards against all relevant reward learning baselines: **LIV** [32] is a robotics reward model pre-trained on EpicKitchens [33], we also fine-tune LIV on $\mathcal{D}_{\text{demos}}$ (**LIV-FT**); **RoboCLIP** [6] uses a pre-trained video language model, S3D [34] trained on HowTo100M [35], to reward agents for language specified tasks; **Video-Language Critic (VLC)** [13] fine-tunes a VLM with a sequential ranking objective to encourage frames later in the video to have higher rewards. We train it on $\mathcal{D}_{\text{demos}}$; **Generative Value Learning (GVL)** [22] prompts a pre-trained Gemini LLM [36] with shuffled frames to predict per-frame progress.

We conduct our primary reward analysis using the simulated Metaworld benchmark [37] because it enables efficient collection of exemplar failed and partially successful rollout videos for analysis. Smaller-scale real-world analyses, **strongly aligned** with simulation, are in Section D-C. For fair comparison, we include a variant of ReWiND trained without $\mathcal{D}_{\text{open-x}}$ (**ReWiND w/o OXE**). Results are evaluated on 17 unseen Metaworld tasks. These tasks are visually similar to training tasks but require new motions to solve (e.g., `Door-Open` → `Door-Close`). We average metrics across 5 demos per task.

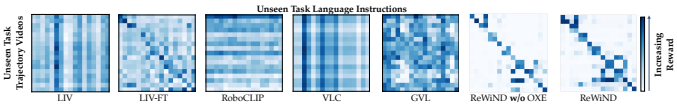



Fig. 4: **Video-Language Reward Confusion Matrix.** For each unseen Metaworld task, we compute rewards for all combinations of demonstration videos and language descriptions. ReWiND produces the most diagonal-heavy confusion matrix, indicating strong alignment between unseen demos and instructions. See Section D-A for train task results, Section D-C for real-world results.

**Generalization.** Next, we evaluate how consistently rewards reflect progress over time in successful, unseen demonstrations. We report Pearson correlation ($r$) of each model's reward against time, and Spearman's rank correlation ($\rho$), which, unlike $r$, captures monotonicity regardless of linearity. As shown in Table I**(a)**, ReWiND again outperforms all baselines—achieving a **30%** relative improvement in $r$ and **27%** in $\rho$ over the best alternative (VLC).

**Policy Rollout Reward Alignment.** We also find that ReWiND can properly reward *failed* policy rollouts, which is important for rewarding RL policies on unseen tasks. For each task, we train an SAC [38] policy from scratch and use trajectories collected from various points of training to construct three evaluation video datasets: `failure`, `near-success`, and `success` containing failed trajectories, trajectories where the policy was close to the goal state but did not succeed, and successful trajectories, respectively. Each task has 2 trajectories of each type.

We evaluate each dataset's relative alignment *ranking* (measured by Spearman's $\rho$) with each reward model. For example, for a given task, if the average reward for a `failure` video is 0.1, a `near-success` video is 0.5, and `success` video is 0.9, then the rankings would be 1, 2, 3, respectively, where 3 corresponds to the best ranking. Thus, $\rho$ over the rankings tells us how often the videos are correctly ranked. We report the ranking $\rho$ in Table I**(b)**. We also report the average *difference* between rewards for `success` with `near-success` and `near-success` with `failure` videos. Overall, likely due to *video rewinding*, ReWiND has a relative **74%** improvement in reward order and **58%** improvement in reward differences over the best baseline, LIV-FT. Additionally, we qualitatively demonstrate how these rankings translate into policy rollout rewards in Appendix Figure 7 by plotting per-frame reward curve predictions of ReWiND against reward baselines for an unsuccessful policy rollout.

**Robustness to Varied Inputs.** Finally, we demonstrate ReWiND's robustness to diverse instructions. For each evaluation task, we manually create three additional language instruc-

[2] Success bonuses can come from a human supervisor [27], learned function [28], or LLM [29]. Our experiments assume a human supervisor because manual resets are required regardless. While we could threshold $R_\psi(o_{1:t}, z)$ outputs to automatically determine success, unseen evaluation task reward ranges can vary, rendering this approach ineffective. Future work could integrate ReWiND with methods reducing human resets [30], [31] and automatic success detectors for truly autonomous RL.

TABLE I: **Combined Evaluation Metrics.** Comparison of reward models across three axes: (1) Demo Video Reward Alignment, (2) Policy Rollout Reward Ranking, and (3) Input Robustness.

| Category | Metric | LIV | LIV-FT | RoboCLIP | VLC | GVL | ReWiND w/o OXE | ReWiND w/ OXE |
|---|---|---|---|---|---|---|---|---|
| **(a) Demo Reward Alignment** | $r \uparrow$ | -0.03 | 0.55 | 0.01 | 0.64 | 0.52 | 0.67 | **0.83** |
| | $\rho \uparrow$ | -0.04 | 0.55 | -0.01 | 0.62 | 0.57 | 0.64 | **0.79** |
| **(b) Policy Rollout Ranking** | Rew. Order $\rho \uparrow$ | -0.32 | 0.47 | 0.00 | -0.18 | 0.32 | 0.76 | **0.82** |
| | Rew. Diff. $\uparrow$ | -0.16 | 0.26 | 0.06 | -0.15 | 0.17 | 0.39 | **0.41** |
| **(c) Input Robustness** | Avg. $\rho \uparrow$ | 0.03 | 0.27 | 0.00 | 0.60 | 0.58 | 0.55 | **0.74** |
| | $\rho$ Variance $\downarrow$ | 0.08 | 0.28 | **0.00** | **0.00** | 0.01 | 0.03 | 0.04 |

tions (without prior knowledge of ReWiND's performance), resulting in four total instructions per task. For example, "close the door" is an original instruction, and we add "shut the door." Each set of instructions is paired with a single demonstration video, and we compare the reward models by measuring their average Spearman's rank correlation ($\rho$) and output variance across these instructions in Table I**(c)**. Higher variance indicates lower robustnes. Again, ReWiND outperforms baselines, achieving the highest average correlation (0.74), **23%** better than VLC, and near-zero variance, even without OXE training—likely aided by our instruction augmentation approach (Section II-A1). RoboCLIP and VLC show near-zero variance but achieve significantly lower correlation scores.

So far, our results demonstrate that **ReWiND significantly outperforms all image-language-conditioned reward baselines** in terms of **generalization**, rewarding **policy rollouts**, and input **robustness**. We next demonstrate how these results translate into sample-efficient policy learning.

### *B. Q2: Learning New Tasks with RL*

**Simulation.** We use the Meta-World simulation benchmark [37], where we pre-train reward models and policies on 20 tasks, each with 5 per-task demos collected from a scripted policy. We evaluate on 8 unseen tasks in Meta-World, chosen for reasonable initial policy rollout behaviors, across 3 seeds each. We compare ReWiND against the 2 language-conditioned reward model baselines that performed best in reward alignment (**VLC**) and policy rollout rankings (**LIV-FT**) from the reward analysis in Section III-A. We also compare against **Sparse**, which pre-trains and fine-tunes on only the sparse success reward bonus, and **Pre-train**, which pre-trains on sparse reward and is evaluated zero-shot on new tasks. All baselines are image, proprioception ($x, y, z$, gripper), and language conditioned. Each method uses the same policy pre-training and RL procedure as ReWiND as outlined in Section II-B, and is trained online for 100k timesteps. See Appendix B for environment and policy training details.

As recommended by Agarwal, Schwarzer, Castro, *et al.* [39], we report the interquartile mean (IQM) and 95% confidence intervals computed over all task success rates at 100k environment steps in Figure 5. Sparse reward fine-tuning and Pre-train (no fine-tuning) result in near-zero success rates, highlighting the difficulty of image-based new task learning under limited data. In fact, Sparse reward fine-tuning, which relies purely on a sparse success bonus, performs worse than Pre-train after fine-tuning. Meanwhile, ReWiND achieves

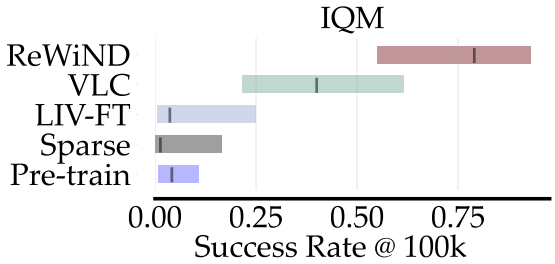



Fig. 5: **Meta-World final performance.** We plot inter-quartile means (IQMs) of success rates after 100k environment steps on 8 unseen tasks in Meta-World. ReWiND achieves 79%.

an IQM success rate of **79%**, a **97.5%** improvement over the best baseline, VLC, demonstrating that ReWiND effectively enables the policy to learn new tasks in Meta-World. These results are well-aligned with our reward analysis in Section III-A, demonstrating how they correlate with policy learning performance. ReWiND is also more sample-efficient at timesteps less than 100k; see extended discussion in Section D-B and sample efficiency curves in Figure 13i.

These results are well-aligned with our reward analysis in Section III-A, demonstrating how they correlate with policy learning performance. ReWiND is also more sample-efficient at timesteps less than 100k.

**Real-World Robot Learning.** We conduct real-world tabletop manipulation experiments with a bimanual Koch v1.1 robot arm setup [40]. We use 5 demos to train the reward function, but 10 for the policy, as we found policy learning to be a bottleneck on this difficult robot embodiment. Across five tasks, we demonstrate in Figure 6 that an hour of real-world RL with ReWiND improves the success rate over the base pre-trained policy from an average 12% success rate to 68%, a **5**$\times$ improvement. Meanwhile, VLC only improves from 8% to 10%—ReWiND outperforms VLC, the best simulation baseline, by **6.7**$\times$. RL for an hour of real-world experiment time corresponds to 50k environment steps with our parallelized codebase that trains the policy while an older checkpoint gathers data in the environment to avoid any training wait time. We select diverse tasks that demonstrate real-world improvement based on generalization metrics defined in prior work [17], [18] on: an in-distribution task, `separate the blue and orange cups`; an in-distribution *difficult* task, `fold the blue towel`; an unseen task in terms of large amounts of *visual* clutter, `open the red trash bin`; an unseen task in terms of spatial relationships between

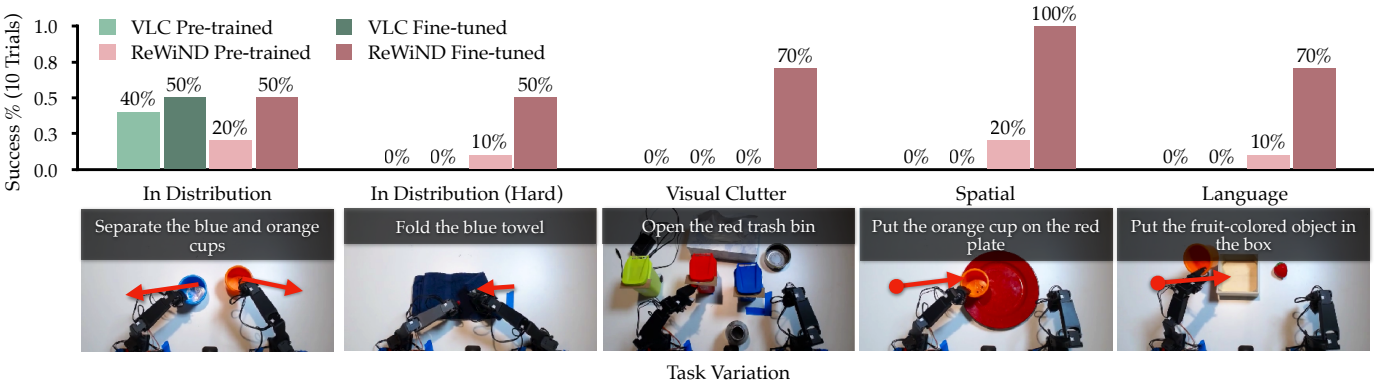



Fig. 6: **Real-robot RL.** We present results on the Koch bimanual arms across in-distribution tasks and visual, spatial, and linguistic generalization tasks. Online RL with ReWiND improves a pre-trained policy by an absolute 56% across all five tasks.

TABLE II: **Ablation Study:** subtracting (−) and adding + various ReWiND components on Metaworld training and evaluation task (a) demo reward alignment, evaluation task (b) policy rollout ranking order and reward difference, and evaluation task (c) input robustness.

| Model | (a) Demo Reward Alignment | | (b) Policy Rollout Ranking | | (c) Input Robustness | |
|---|---|---|---|---|---|---|
| | Train Demos $\rho \uparrow$ | Unseen Demo $\rho \uparrow$ | Rew. Order $\rho \uparrow$ | Rew. Diff. $\uparrow$ | Avg. $\rho \uparrow$ | $\rho$ Variance $\downarrow$ |
| **Original ReWiND** | **1.00** | 0.79 | **0.82** | **0.41** | 0.74 | 0.04 |
| − **Targ. Env Data** | 0.55 | 0.77 | 0.18 | 0.08 | **0.78** | 0.04 |
| − **Open-X Subset** | **1.00** | 0.64 | 0.76 | 0.39 | 0.55 | 0.03 |
| − **Video Rewind** | **1.00** | 0.69 | 0.56 | 0.27 | 0.66 | **0.02** |
| − **Instr. Generation** | **1.00** | 0.66 | 0.62 | 0.30 | 0.52 | 0.07 |
| + **Full Pos. Embeds** | 0.99 | **0.85** | 0.71 | 0.33 | **0.78** | 0.06 |

objects requiring *new action sequences*, `put the orange cup on the red plate`; and an unseen task in terms of *language* input, `put the fruit-colored object in the box`. Overall, ReWiND enables real-world reinforcement learning on unseen tasks without requiring new demonstrations, improving over the pretrained pre-trained policy, and outperforms the best baseline from simulation, VLC.

## IV. Ablation Study and Additional Analysis

### A. Ablation Study

We perform a thorough ablation study of ReWiND regarding how specific design choices influence demonstration reward alignment, policy rollout ranking, and input robustness metrics introduced in Section III-A. We ablate: instruction generation and video rewinding (Section II-A1); using OXE data; the need for target environment data $\mathcal{D}_{\text{demos}}$; and finally, the use of first frame vs. full frame positional embeddings on the input observation sequence $o_{1:T}$ in the cross-modal sequential aggregator (Section II-A2). Overall, the original ReWiND model performs best across most metrics. Below, we analyze the impact of each ablation:

**Datasets.** Removing target environment data (−**Targ. Env Data**)—i.e., using only $\mathcal{D}_{\text{open-x}}$ data without $\mathcal{D}_{\text{demos}}$—leads to poor alignment with training demonstrations (Table IIa) and fails to distinguish between failed, near-successful, and successful policy rollouts (Table IIb). However, it retains strong input robustness due to the diversity of OXE data. Meanwhile, removing the Open-X subset (−**Open-X Subset**) harms unseen task reward alignment (Table IIa) and input robustness (Table IIc), highlighting the importance of OXE data for generalizing across varied language instructions.

**Augmentation.** Eliminating video rewinding (−**Video Rewind**) degrades rollout ranking performance (Table IIb), showing that rewinding helps distinguish failed rollouts as intended. This variant performs similarly to the single-image LIV-FT baseline in Table I, indicating that video rewinding more effectively captures the temporal information in the videos. Similarly, removing instruction generation (−**Instruction Generation**) reduces performance on language input robustness (Table IIc), confirming that LLM-generated instructions enhance robustness to diverse inputs.

**Architecture.** Adding full positional embeddings (+**Full Pos. Embeds**) improves unseen demo alignment (Table IIa) but worsens rollout ranking (Table IIb), likely due to overfitting—where the model learns to predict increasing rewards regardless of input. To avoid overfitting, the main ReWiND model uses only first-frame positional embeddings (Section II-A2).

**Concluding Statement.** In conclusion, our experiments demonstrated ReWiND's effectiveness as a reward function for policy learning through detailed reward analyses and its effectiveness as a framework for sample-efficient robot learning of unseen tasks, both in simulation and on a real bimanual robot.

## References


[1] A. Brohan *et al.*, "Rt-1: Robotics transformer for real-world control at scale," in *Robotics: Science and Systems (RSS)*, 2023.

[2] A. Brohan *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning (CoRL)*, 2023.

[3] K. Black *et al.*, "$\pi_0$: A vision-language-action flow model for general robot control," *arXiv preprint arxiv:2410.24164*, 2024. arXiv: 2410.24164 [cs.LG].

[4] Y. Li *et al.*, "Hamster: Hierarchical action models for open-world robot manipulation," in *International Conference on Learning Representations (ICLR)*, 2025.

[5] R. S. Sutton *et al.*, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018, ISBN: 0262039249.

[6] S. A. Sontakke *et al.*, "Roboclip: One demonstration is enough to learn robot policies," in *NeurIPS*, 2023.

[7] M. Kwon *et al.*, "Reward design with language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[8] H. Hu *et al.*, "Language instructed reinforcement learning for human-ai coordination," in *International Conference on Machine Learning (ICML)*, 2023.

[9] W. Yu *et al.*, "Language to rewards for robotic skill synthesis," in *Conference on Robot Learning (CoRL)*, 2023.

[10] Y. J. Ma *et al.*, "Eureka: Human-level reward design via coding large language models," in *International Conference on Learning Representations (ICLR)*, 2024.

[11] Y. J. Ma *et al.*, "Dreureka: Language model guided sim-to-real transfer," in *Robotics: Science and Systems (RSS)*, 2024.

[12] W. Liang *et al.*, "Environment curriculum generation via large language models," in *Conference on Robot Learning (CoRL)*, 2024.

[13] M. Alakuijala *et al.*, "Video-language critic: Transferable reward functions for language-conditioned robotics," in *Transactions on Machine Learning Research (TMLR)*, 2025.

[14] Y. Wang *et al.*, "Rl-vlm-f: Reinforcement learning from vision language foundation model feedback," in *International Conference on Machine Learning (ICML)*, 2024.

[15] Z. Yang *et al.*, "Trajectory improvement and reward learning from comparative language feedback," in *Conference on Robot Learning (CoRL)*, 2024.

[16] O. X.-E. Collaboration *et al.*, "Open X-Embodiment: Robotic learning datasets and RT-X models," in *International Conference on Robotics and Automation (ICRA)*, 2024.

[17] A. Anwar *et al.*, "Contrast sets for evaluating language-guided robot policies," in *Conference on Robot Learning (CoRL)*, 2024.

[18] J. Gao *et al.*, "A taxonomy for evaluating generalist robot policies," *arXiv preprint arXiv:2503.01238*, 2025.

[19] D. Yang *et al.*, "Rank2reward: Learning shaped reward functions from passive video," in *International Conference on Robotics and Automation (ICRA)*, 2024.

[20] M. Oquab *et al.*, *Dinov2: Learning robust visual features without supervision*, 2024. arXiv: 2304.07193 [cs.CV].

[21] N. Reimers *et al.*, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[22] Y. J. Ma *et al.*, "Vision language models are in-context value learners," in *International Conference on Learning Representations (ICLR)*, 2025.

[23] I. Kostrikov *et al.*, "Offline reinforcement learning with implicit q-learning," in *International Conference on Learning Representations (ICLR)*, 2022.

[24] S. Venkataraman *et al.*, "Real-world offline reinforcement learning from vision language model feedback," in *arXiv preprint arXiv:2411.05273*, 2024.

[25] J. Zhang *et al.*, "Bootstrap your own skills: Learning to solve new tasks with large language model guidance," in *Conference on Robot Learning (CoRL)*, 2023.

[26] J. Zhang *et al.*, "Sprint: Scalable policy pre-training via language instruction relabeling," in *International Conference on Robotics and Automation (ICRA)*, 2024.

[27] J. Luo *et al.*, "Precise and dexterous robotic manipulation via human-in-the-loop reinforcement learning," in *arXiv preprint arXiv:2410.21845*, 2024. arXiv: 2410.21845 [cs.RO].

[28] J. Fu *et al.*, "Variational inverse control with events: A general framework for data-driven reward definition," in *NeurIPS*, S. Bengio *et al.*, Eds., 2018.

[29] W. Ye *et al.*, "Reinforcement learning with foundation priors: Let embodied agent efficiently learn on its own," in *Conference on Robot Learning (CoRL)*, 2024.

[30] J. Yang *et al.*, "Robot fine-tuning made easy: Pre-training rewards and policies for autonomous real-world reinforcement learning," in *International Conference on Robotics and Automation (ICRA)*, 2024.

[31] A. Gupta *et al.*, "Reset-free reinforcement learning via multi-task learning: Learning dexterous manipulation behaviors without human intervention," in *International Conference on Robotics and Automation (ICRA)*, 2021.

[32] Y. J. Ma *et al.*, "Liv: Language-image representations and rewards for robotic control," in *International Conference on Machine Learning (ICML)*, 2023.

[33] D. Damen *et al.*, "Rescaling egocentric vision: Collection, pipeline and challenges for epic-kitchens-100," *International Journal of Computer Vision (IJCV)*, vol. 130, pp. 33–55, 2022.

[34] S. Xie *et al.*, "Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification," in *European Conference on Computer Vision (ECCV)*, 2018.

[35] A. Miech *et al.*, "Howto100m: Learning a text-video embedding by watching hundred million narrated video clips," in *International Conference on Computer Vision (ICCV)*, 2019.

[36] G. Team, "Gemini: A family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2024. arXiv: 2312.11805 [cs.CL].

[37] T. Yu *et al.*, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2019.

[38] T. Haarnoja *et al.*, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)*, 2018.

[39] R. Agarwal *et al.*, "Deep reinforcement learning at the edge of the statistical precipice," in *NeurIPS*, 2021.

[40] R. Cadene *et al.*, *Lerobot: State-of-the-art machine learning for real-world robotics in pytorch*, https://github.com/huggingface/lerobot, 2024.

[41] H. R. Walke *et al.*, "Bridgedata v2: A dataset for robot learning at scale," in *Conference on Robot Learning (CoRL)*, 2023.

[42] E. Jang *et al.*, "BC-z: Zero-shot task generalization with robotic imitation learning," in *Conference on Robot Learning (CoRL)*, 2021.

[43] S. Dass *et al.*, *Clvr jaco play dataset*, version 1.0.0, 2023. [Online]. Available: https://github.com/clvrai/clvr_jaco_play_dataset.

[44] L. Y. Chen *et al.*, *Berkeley UR5 demonstration dataset*, https://sites.google.com/view/berkeley-ur5/home, 2023.

[45] X. Zhu *et al.*, *Fanuc manipulation: A dataset for learning-based manipulation with fanuc mate 200id robot*, https://sites.google.com/berkeley.edu/fanuc-manipulation, 2023.

[46] S. Bahl *et al.*, "Affordances from human videos as a versatile representation for robotics," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[47] R. Mendonca *et al.*, "Structured world models from human videos," *Conference on Robot Learning (CoRL)*, 2023.

[48] S. Belkhale *et al.*, "Hydra: Hybrid robot actions for imitation learning," in *Conference on Robot Learning (CoRL)*, 2023.

[49] G. Yan *et al.*, *Ucsd kitchens dataset*, Aug. 2023.

[50] Y. Zhu *et al.*, "Bottom-up skill discovery from unsegmented demonstrations for long-horizon robot manipulation," *IEEE Robotics and Automation Letters (RA-L)*, 2022.

[51] H. Liu *et al.*, "Robot learning on the job: Human-in-the-loop autonomy and learning during deployment," in *Robotics: Science and Systems (RSS)*, 2023.

[52] J. Wu *et al.*, *V-former: Offline RL with temporally-extended actions*, 2024.

[53] X. B. Peng *et al.*, "Advantage-weighted regression: Simple and scalable off-policy reinforcement learning," in *arXiv preprint arXiv:1910.00177*, 2019.

[54] Z. Zhou *et al.*, "Efficient online reinforcement learning fine-tuning need not retain offline data," in *International Conference on Learning Representations (ICLR)*, 2025.

[55] P. J. Ball *et al.*, "Efficient online reinforcement learning with offline data," in *International Conference on Machine Learning (ICML)*, 2023.

[56] I. Uchendu *et al.*, "Jump-start reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2023.

[57] A. Q. Jiang *et al.*, "Mistral 7b," *arXiv preprint arXiv:2401.04088*, 2023. arXiv: 2310.06825 [cs.CL].

[58] T. Z. Zhao *et al.*, "Learning fine-grained bimanual manipulation with low-cost hardware," *arXiv preprint arXiv:2304.13705*, 2023.

[59] K. Pertsch *et al.*, "Accelerating reinforcement learning with learned skill priors," in *Conference on Robot Learning (CoRL)*, 2020.

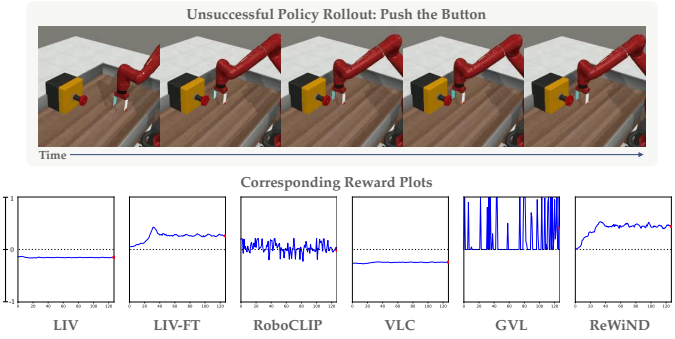



Fig. 7: Unsuccessful policy rollout for the "Push the Button" task in Meta-World and its corresponding rewards below it. ReWiND predicts calibrated rewards that reflect better partial progress when the policy gets stuck near the button.

## APPENDIX A
## IMPLEMENTATION DETAILS

This section introduces implementation details for ReWiND in terms of the datasets, reward model, policy training, and online RL.

### A. ReWiND Implementation

Full pseudocode for ReWiND is listed in Algorithm 1. Individual implementation details follow.

*1) Open-X Dataset:* Below we list details of the OXE subset, $\mathcal{D}_{\text{open-x}}$, used for training the reward model $R_\psi(o_{1:t}, z)$ (mentioned in Section II-A).

We select a subset of datasets from the Open-X Dataset [16]. The subset includes Bridge-V2 [41], BC-Z [42], Fractal [1], CLVR Jaco Play [43], Berkeley Autolab UR5 [44], Berkeley Fanuc Manipulation [45], CMU Stretch [46], [47], Stanford Hydra [48], UCSD Kitchen [49], Austin BUDS [50], and Austin Sirius [51]. These datasets were selected for their high-quality, task-oriented manipulation trajectories (i.e., no play data or extremely high-level annotations). These datasets provide around 350k trajectories and 58k total unique task annotations. To ensure meaningful trajectories for training the ReWiND reward model, we postprocess the data to remove trajectories with less than 5 timesteps. We subsample the videos in the datasets to 16 frames for reward model training, as we did not see a noticeable benefit from training it with longer videos.

*2) Reward Function:* We picture the overall architecture of the reward function in Figure 8. We encode input images with the pre-trained DINO-V2 base model (86M params) with 768 embedding size. Similarly, we encode language with the pre-trained ALL-MINILM-L12-V2 model with a 384 embedding size. We project image and language embeddings to 512 dimensions with a single linear layer. We treat the language embedding as a single input token and we evenly downsample DINO-v2 image embeddings for every observation sequence to 16 frames.

The cross-modal sequential aggregator takes these tokens as input and produces a per-image embedding used by an MLP

**Algorithm 1** ReWiND Algorithm, Section I.

**Require:** Demo dataset $\mathcal{D}_{\text{demos}}$, Pre-trained LLM, Open-X subset $\mathcal{D}_{\text{open-x}}$, Reward Model $R_\psi(o_{1:t}, z)$, Policy $\pi$. $\mathcal{D}_{\text{demos}}$ includes video trajectories $o_{1:t}$ and language embedding $z$.

/* *Train the Reward Model* Section II-A */
REWARDMODELTRAINING($R_\psi(o_{1:t}, z)$, $\mathcal{D}_{\text{demos}}$, $\mathcal{D}_{\text{open-x}}$)
/* *Policy Pretraining* Section II-B */
OFFLINEPOLICYPRETRAINING($R_\psi(o_{1:t}, z)$, $\mathcal{D}_{\text{demos}}$, $\pi$)
/* *Learn New Task Online* Section II-B */
ONLINERL($z_{\text{new}}$, $R_\psi(o_{1:t}, z)$, $\pi$ )

**procedure** REWARDMODELTRAINING($R_\psi(o_{1:t}, z)$, $\mathcal{D}_{\text{demos}}$, $\mathcal{D}_{\text{open-x}}$)
    Augment instruction labels with LLM
    Sample a video clip and annotation $o_{t_1:t_2}, z$ from $\mathcal{D}_{\text{demos}}$ or $\mathcal{D}_{\text{open-x}}$.
    Choose to keep the original video or perform REWINDAUGMENTATION.
    **if** perform REWINDAUGMENTATION **then**
        $o^{\text{rewound}} \leftarrow$ REWINDAUGMENTATION($o_{t_1:t_2}$)
        Optimize $R_\psi(o_{1:t}, z)$ with $\mathcal{L}_{\text{rewind}}(o^{\text{rewound}}, z)$ ▷ Equation (2)
    **else**
        Sample a different video clip $o^{\text{other}}_{t_1':t_2'}$
        Optimize $R_\psi(o_{1:t}, z)$ with $\mathcal{L}_{\text{progress}}(o_{t_1:t_2}, z, o^{\text{other}}_{t_1',t_2'})$ ▷ Equation (1)
    **end if**
**end procedure**

**procedure** OFFLINEPOLICYPRETRAINING($R_\psi(o_{1:t}, z)$, $\mathcal{D}_{\text{demos}}$, $\pi$)
    Relabel $\mathcal{D}_{\text{demos}}$ with $\hat{r}^{\text{off}}$ coming from $R_\psi(o_{1:t}, z)$. ▷ Equation (4)
    Train $\pi$ with offline RL on relabeled $\mathcal{D}_{\text{demos}}$.
**end procedure**

**procedure** ONLINERL($R_\psi(o_{1:t}, z)$, $\pi$)
    For every rollout label the trajectories with $\hat{r}^{\text{on}}$ from $R_\psi(o_{1:t}, z)$. ▷ Equation (5)
    Optimize $\pi$ with online RL Algorithm
**end procedure**

**procedure** REWINDAUGMENTATION($o_{t_1:t_2}$) ▷ Section II-A1
    Sample random split point $i$ between $t_1$ and $t_2$.
    Sample # frames to rewind for, $k$
    Reverse $o_{i-k:i}$ and concat with $o_{t_1:i}$
    Return $[o_{t_1:i-1}, o_{i:i-k}]$
**end procedure**

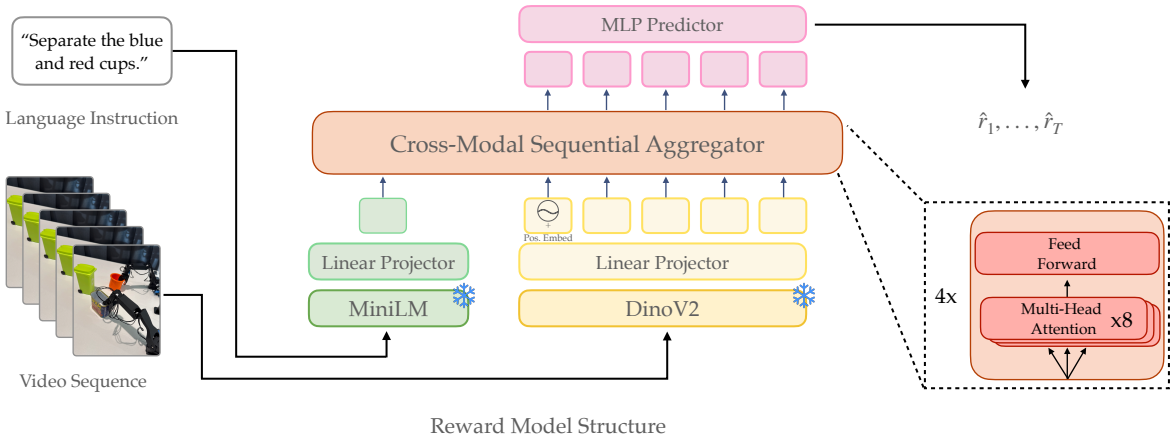



Fig. 8: **ReWiND's Reward Model Architecture**. It's composed of frozen language and image input embeddings projected to a shared hidden dimension of 512. These embeddings are treated as input tokens to the cross-modal sequential aggregator transformer composed of 4 causally masked transformer layers composed of 8 multi-head attention blocks each. Per-timestep embeddings for each input observation are fed into an MLP to predict rewards for each timestep.

to produce per-timestep rewards. The cross-modal sequential aggregator is a causally masked transformer (`PyTorch nn.TransformerEncoder`) composed of 4 layers, each with 8 heads with a combined hidden dimension of 2048. We add a learnable positional embedding to only the first frame of the video sequence embedding. In the ReWind reward function training phase, we trained 2k steps for Meta-World and 10k steps for Real-World robot experiments, with a batch size of 1024. Each batch includes 80% data from $\mathcal{D}_{\text{open-x}}$ and 20% target environment data from $\mathcal{D}_{\text{demos}}$. Each video in the batch has an 80% probability of having video rewind augmentation, and independently, a 20% percent probability of having a mismatched video-language pairing with 0 progress target (see Section II-A). In order to better policy execution videos that look *close to success*, 10% of the rewound videos will only have their last 3 frames rewound. No extensive tuning was performed on these per-sample rewind and mismatch probabilities; they were heuristically chosen during initial small-scale experimentation and then fixed for all experiments.

*3) Policy Training:* Specific architectural and training details are discussed per-environment in the corresponding sections Section B-B and Section C-B. Below we talk about high level algorithmic details for policy training along with shared implementation details across environments.

*a) Policy Input.:* Similar to the reward model, we condition the policy on frozen pre-trained image and language embeddings: DINO-v2-base image embeddings (86M params, 768 embedding size) [20] along with ALL-MINILM-L12-V2 language embeddings of size 384 from the Sentence Transformers python package [21]. We also include proprioceptive information in both of our envrionments.

*b) Offline RL.:* We use Implicit Q Learning (IQL) [23] as prior work found it performant and easy to tune for robot manipulation with action-chunked policies [25], [26], [52]. IQL trains on in-distribution $(s, a, s', r, a')$ tuples from the dataset, avoiding using next actions $a'$ sampled from a policy, to ensure the critic functions accurately reflect returns restricted to dataset actions. The value function is optimized with expectile regression, controlled by a hyperparameter $\tau$: $\tau = 0.5$ recovers mean squared error, while $\tau \to 1$ yields a more optimistic estimate, helping the value function "stitch" together distant rewards in sparse settings. The policy is trained via advantage-weighted regression [53], maximizing

$$e^{\beta(Q(s,a)-V(s))} \log \pi(a|s),$$

where $\beta$ is a temperature hyperparameter controlling how "spiky" the policy loss is. To prevent numerical instability, the exponential term is capped at a maximum value in practice (for us, this is 100).

*c) Online RL.:* We use a custom soft-actor critic (SAC) [38] implementation initialized with the pre-trained policy from offline RL along with the Q and target Q functions. We follow best practices from recent offline-online RL fine-tuning work [54], [55], namely:

- 5-10 critics instead of 2, with random sampling of critics
- LayerNorm in the critic and possibly LayerNorm in the policy
- A higher update-to-data ratio in the critics
- "Warm-starting" online RL by running with the frozen pre-trained policy for the first few thousand environment steps [54], [56]
- Possibly sampling offline pre-training data at a 50% ratio during online RL
- Removing the SAC entropy term from the target critic

We found that by default, efficient offline-online learning algorithms did not work very well "out of the box" for learning *new* tasks on our real robot. This is perhaps because they focus specifically on offline-online fine-tuning on the same task while we are trying to learn new tasks, or perhaps due to additional challenges of real-robot RL. Therefore, we make some per-environment design decisions for online RL detailed in the respective environment training sections.

## APPENDIX B
## METAWORLD EXPERIMENTS

### A. Simulation Setup

*a) Training/Eval Task Selection.:* We manually select 20 training tasks from MT50 benchmark in the Metaworld environment. These tasks are used for both reward model training and policy pre-training. The training tasks include: `Button-Press`, `Button-Press-Topdown-Wall`, `Coffee-Pull`, `Dial-Turn`, `Door-Open`, `Door-Unlock`, `Drawer-Close`, `Faucet-Open`, `Handle-Press`, `Handle-Pull-Side`, `Peg-Insert-Side`, `Pick-Place`, `Plate-Slide`, `Plate-Slide-Back-Side`, `Push`, `Reach`, `Stick-Push`, `Stick-Pull`, `Window-Open`, `Hand-Insert`.

We also choose another 17 tasks from the MT50 benchmark for reward model evaluation and 8 of tasks are selected for downstream policy finetuning.[3] The evaluation tasks include `Window-Close`, `Sweep-Into`, `Soccer`, `Reach-Wall`, `Push-Back`, `Plate-Slide-Side`, `Plate-Slide-Back`, `Pick-Place-Wall`, `Handle-Pull`, `Handle-Press-Side`, `Faucet-Close`, `Door-Lock`, `Door-Close`, `Coffee-Push`, `Coffee-Button`, `Button-Press-Wall`, `Button-Press-Topdown`. These tasks are visually similar to the training tasks, but the tasks are different. The 8 tasks used for downstream policy training are `Window-Close`, `Reach-Wall`, `Handle-Pull`, `Coffee-Button`, `Button-Press-Wall`, `Door-Lock`, `Handle-Press-Side`, `Sweep-into`.

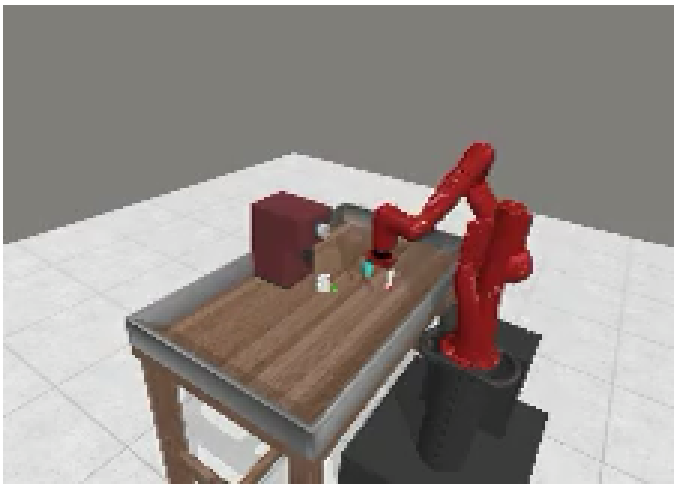


Fig. 9: Example camera viewpoint in Metaworld.

*b) Environment Details.:* We use Metaworld [37] with the default 3rd-person camera viewpoint, pictured in Figure 9, and also 4-dimension proprioception input $(x, y, z, \text{gripper})$. The policy action space is the default one from Metaworld represented as a 4-dimensional relative action space for $(\Delta x, \Delta y, \Delta z, \text{gripper})$. Unlike the Metaworld environment setups in prior reward learning papers, we do not include goal/ground truth state information. We also terminate the environment on success. Both of these choices were made to mimic a real-world robot learning setup. The time horizon of each episode is limited to 128 steps. The success bonus for online and offline RL used in Equation (4) and Equation (5) is 200 for ReWiND and all baselines.

[3]These 17 tasks were chosen for sharing at least some characteristic with a training task.


Fig. 10: **Real World Bimanual Robot Setup.** Our real-world setup consists of a top-down and side camera mounted to a table where two Koch v1.1 low-cost arms are mounted. This setup allows us to perform bimanual tasks and easily collect data with another pair of low-cost "leader" arms mounted to the same table.

### B. Training Details

For $\mathcal{D}_{\text{demos}}$, we select 20 tasks from the MT-50 benchmark. Each task consists of one human-labeled annotation, four augmented annotations (Section II-A1), and five optimal demonstrations produced by the MetaWorld built-in planner. We render images at the default resolution of 640x480, center-crop to 224x224 and embed the image with DINOv2 encoder.

We pre-train the policy with IQL [23] for 100K steps with learning rate 0.001, gamma 0.99. We use a three layer MLP of size $[768, 512, 256]$ for both the policy and value function network. The general training procedure is described in (Section A-A3)

For the various hyperparameters for online policy learning we used in MetaWorld as described in Section A-A3. We use 10 critics and sample 2 of them during training, LayerNorm in both the critic and policy, and an update-to-data ratio of 4 for the critics. We are not sampling from offline pre-training data during online training nor are we training the target critic with the entropy term, so the implementation is identical to Warm-Start RL [54]. We warm-start online RL for 4000 steps.

## APPENDIX C
## REAL ROBOT EXPERIMENTS

### A. Robot Experiment Setup

We use the Koch1.1 bimanual arm setup for data collection and learning [40].[4] Altogether, four total arms (2 for data collection) cost ∼$1000, letting us demonstrate ReWiND enables real-world online RL of new tasks even with very low-cost

[4]https://github.com/jess-moss/koch-v1-1

hardware and noisy control. The observations consist of RGB images from a Logitech C930e top camera and side camera (pictured in Figure 10). We control the robot with absolute joint position control at a frequency of 30Hz. We collect a small dataset of 10 demonstrations over 20 tasks, and then use 5 demos per-task for the reward function. We found the offline-trained policy to be the primary bottleneck to optimizing rewards in unseen tasks, so we used 10 demos per-task for offline policy training. We have an episode timeout of 250 steps and provide a success bonus of 125 upon success (from Equation (4) and Equation (5)). Proprioceptive information in this environment includes 12 robot joint states, 6 for each arm. These represent the rotation of each joint and gripper..

### *B. Real Robot Training Details*

We use a small, instruction-tuned, open-source LLM, `Mistral-7B-Instruct-v0.3` [57], to generate 9 additional instructions for each task for instruction augmentation.

For the small dataset in real robot experiments, we manually choose 15 tasks in the Koch tabletop setting, and each task includes 5 trajectories and 10 annotations. The evaluation set is 5 other random tasks, which are irrelevant with the tasks in the small dataset. We use this evaluation set for offline metrics and validating various design choices.

Unlike the MetaWorld experiments that use an MLP-based policy, we use an action-chunked policy with temporal ensembling for the real robot. We found chunking to lead to more stable bimanual manipulation on the Koch arms. We implement the action chunking with a Transformer policy that predicts 60 actions at each timesteps corresponding to 2 seconds of actions. We also implement a Transformer-based critic. During rollouts, we then use temporal ensembling [58]. Here, the current action is ensembles with the last 60 timesteps' predictions according to an exponential weighting scheme $w_i = \exp(-m*i)$, where we use $m = 0.01$ or $m = 0.1$ depending on the task. We found $m = 0.1$ to work well for tasks requiring grasping solid objects as it weights recent actions more heavily, necessary for ensuring the policy actually commits to the grasp, and $m = 0.01$ to work well for non-grasping tasks as it results in a smoother policy.

The policy is a Transformer decoder with 1 layer and 8 heads with 1.5M params. The critic is a Transformer encoder with 8 heads and 1 layer. We train each policy for 20k steps offline on our offline dataset using IQL with AWR for policy extraction. We train using a batch size of 256, use 5 critics, and subsample 2 critics at each training step. We use LayerNorm only in the critics as we found that LayerNorm in the action-chunked policy could potentially hurt RL performance. We also warm-start online RL for 3000 steps. We do not sample actions during policy rollouts as we found action sampling to conflict with temporal ensembling.

Then, we train the policy online as described in Section II-B. We train online for 50k environment steps, which takes approximately 1 hour as there is minimal waiting time for policy training due to a threaded implementation that trains the policy while the last iteration's policy checkpoint is used for rollouts. This parallelization nearly doubles the rate at which we are able to collect policy rollouts. Specifically, during online training, we collect a single rollout corresponding to 250 environment steps while simultaneously training the policy for 75 gradient steps. We keep a relatively low policy to environment update ratio in order to ensure that we do not have to wait for offline training to finish in order to start the next online rollout. At each gradient step, we sample our buffer such that 50% is the offline training data, 25% is online failure trajectories, and 25% is online successful trajectories. This sampling approach helps upsample successful online trajectories. For every actor gradient step, we do 5 critic update steps to more quickly train the critic online.

During real-world policy rollouts, it is important for the robot to take safe actions that will not crash into other objects or the table. However, we found that when regularizing the policy's KL divergence against a max-entropy prior as is the case in the entropy maximization objective in standard SAC [38], the growing entropy term would cause the policy to produce largely random actions. Therefore, we regularize against the pretrained policy's distribution to encourage reasonable behaviors throughout the process of learning, similar to the SAC update rules from Pertsch, Lee, and Lim [59]. Thus the $\pi$ and $Q$ updates follow:

$$\pi \leftarrow \max_{\pi} \mathbb{E}_{\pi}\Big[Q(o,z) - \alpha \underbrace{\mathrm{KL}(\pi(\cdot|o,z) \;||\; \pi_{\text{pretrained}}(\cdot|o,z)}_{\text{pre-trained policy guidance}}\Big] \tag{6}$$

$$Q \leftarrow \min_{Q} Q(o,z) = r + \gamma\Big[Q(s',z',d') - \alpha \underbrace{\mathrm{KL}\left(\pi(\cdot|o,z) \;||\; \pi_{\text{pretrained}}(\cdot|o,z)\right)}_{\text{pre-trained policy guidance}}\Big] \tag{7}$$

We set $\alpha$ in both equations to a fixed value of 10.0 on tasks where grasping solid objects is not required. For others, we set it to 20.0 to ensure the policy doesn't degenerate from its grasping action early in training. We found that lower KL penalties could result in the policy falling into locally optimal but globally suboptimal behaviors, such as moving a cup with the arm instead of actually picking it up.

### *C. Real Robot Tasks*

We collected 10 demos per-task over 20 tasks on the Koch arms. We train the reward function on 5 demos per-task and the policy on 10 demos per-task. We list these training tasks below.

`Move the orange cup from the left to the right`, `Move the orange cup from the right to the left`, `Put the orange cup on the red plate`, `Put the red cup on the red plate`, `Separate the blue and red cups`, `Fold the blue towel`, `Open the green trash bin`, `Open the blue trash bin`, `Throw the banana away in the green trash bin`, `Throw the banana away in the blue trash bin`, `Put the red marker in the red trash can`, `Put the pink`

`marker in the green trash can`, `Put the blue tape in the box on the left`, `Put the banana in the box`, `Put the orange cup in the box`, `Put the blue cup on the red plate`, `Separate the orange and blue cups`, `Open the red trash bin`, `Throw the banana away in the red trash bin`, `Put the red tape in the box on the right`.

In addition, we present rollouts of the five online tasks in Figure 11. We also provide additional descriptions of these tasks below:

- `Separate the blue and orange cups`: the robot must separate the two cups in the middle
- `Fold the blue towel`: the robot must fold the towel in half.
- `Open the red trash bin`: the robot is surrounded by clutter compared to the training data above and must open the trash bin
- `Put the orange cup in the red plate`: the robot picks an orange cup and must place it on a plate that is further away from the training data distribution
- `Put fruit-colored object in the box`: we refer to a "fruit-colored" object to test the robot's ability to handle semantic generalization.

## APPENDIX D
## ADDITIONAL RESULTS

### A. Additional Metaworld Reward Analysis

In Figure 12 we plot the confusion matrices of different reward models on training tasks in addition to the evaluation task plots of Metaworld in Figure 4. LIV, RoboCLIP and GVL are not pretrained or fine-tuned on the etraining tasks while VLC, LIV-FT and ReWiND are. We can see both ReWiND w/ OXE data $\mathcal{D}_{\text{open-x}}$ and ReWiND w/o OXE data $\mathcal{D}_{\text{open-x}}$ are the best, having the clearest disparity between the diagonal and off-diagonal elements. LIV-FT also works well with a diagonal-heavy matrix. However, its disparity is not as clear as ReWiND.

### B. MetaWorld Sample Efficiency Results

In this section, we analyze the sample efficiency of ReWiND against baselines in Metaworld. Figure 13 plots the learning curves for all downstream policy training tasks. Each panel corresponds to one specific task. And Figure 13i displays the average of all 8 downstream tasks we used for policy fine-tuning. We can see from the average IQM plot that ReWiND achieves higher success rate than other baselines with the same number of timesteps and ReWiND is generally more sample-efficient at any timestep.

### C. Real-World Reward Analysis

We evaluated the performance of ReWiND in Metaworld in Section III-A. In this section, we analyze how ReWiND works with real-world data. For the real-world setup, we use both views of each trajectory, treated as separate videos (but from the same demonstration) to train and evaluate all models.

TABLE III: **Evaluation Metrics on Real-world Unseen Tasks:** Comparsion between reward models in real-world unseen tasks with rank correlation $\rho$ and $r$.

| Model | LIV | LIV-FT | RoboCLIP | VLC | GVL | ReWiND |
|---|---|---|---|---|---|---|
| $\rho \uparrow$ | 0.22 | -0.18 | 0.04 | 0.20 | 0.57 | **0.91** |
| $r \uparrow$ | 0.23 | -0.13 | 0.04 | 0.19 | 0.52 | **0.91** |

It can be seen from Table III that ReWiND has the highest Spearman's rank correlation ($\rho$) and Pearson's rank correlation ($r$) among all reward models. Also, in Figure 14 and Figure 15, ReWiND has the best alignment between true-paired video and language instruction in both training tasks and unseen tasks, displaying strong generalization in new tasks. Note that LIV, GVL, and RoboCLIP are not trained on these training tasks as they are zero-shot models.

## APPENDIX E
## ABLATION STUDY AND ADDITIONAL ANALYSIS

### A. Ablation Study

We perform a thorough ablation study of ReWiND regarding how specific design choices influence demonstration reward alignment, policy rollout ranking, and input robustness metrics introduced in Section III-A. We ablate: instruction generation and video rewinding (Section II-A1); using OXE data; the need for target environment data $\mathcal{D}_{\text{demos}}$; and finally, the use of first frame vs. full frame positional embeddings on the input observation sequence $o_{1:T}$ in the cross-modal sequential aggregator (Section II-A2). Overall, the original ReWiND model performs best across most metrics. Below, we analyze the impact of each ablation:

**Datasets.** Removing target environment data ($-$**Targ. Env Data**)—i.e., using only $\mathcal{D}_{\text{open-x}}$ data without $\mathcal{D}_{\text{demos}}$—leads to poor alignment with training demonstrations (Table IIa) and fails to distinguish between failed, near-successful, and successful policy rollouts (Table IIb). However, it retains strong input robustness due to the diversity of OXE data. Meanwhile, removing the Open-X subset ($-$**Open-X Subset**) harms unseen task reward alignment (Table IIa) and input robustness (Table IIc), highlighting the importance of OXE data for generalizing across varied language instructions.

**Augmentation.** Eliminating video rewinding ($-$**Video Rewind**) degrades rollout ranking performance (Table IIb), showing that rewinding helps distinguish failed rollouts as intended. This variant performs similarly to the single-image LIV-FT baseline in Table I, indicating that video rewinding more effectively captures the temporal information in the videos. Similarly, removing instruction generation ($-$**Instruction Generation**) reduces performance on language input robustness (Table IIc), confirming that LLM-generated instructions enhance robustness to diverse inputs.

**Architecture.** Adding full positional embeddings ($+$**Full Pos. Embeds**) improves unseen demo alignment (Table IIa) but worsens rollout ranking (Table IIb), likely due to overfitting—where the model learns to predict increasing

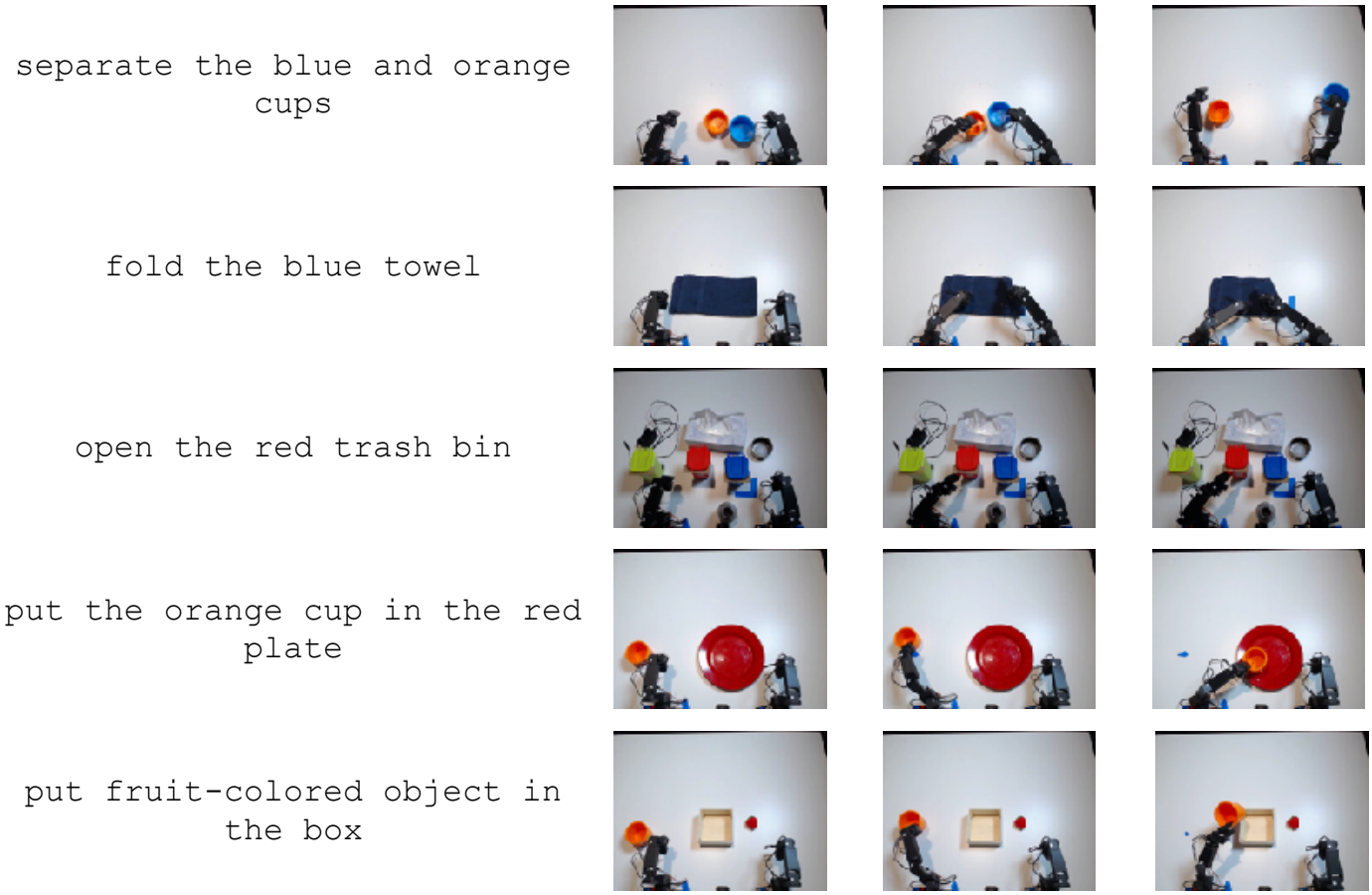



Fig. 11: We present rollouts for the 5 tasks we use for online RL. The first two tasks are in-distribution to the policy, while the latter 3 tasks are out-of-distribution with respect to visual, spatial, or semantic generalization.

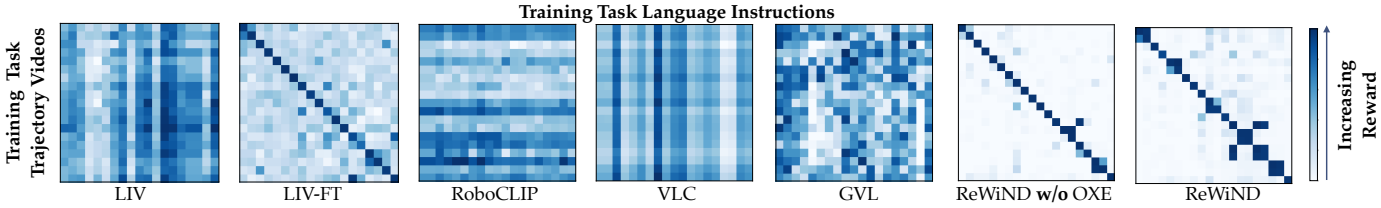
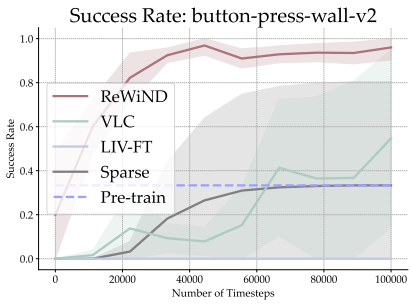
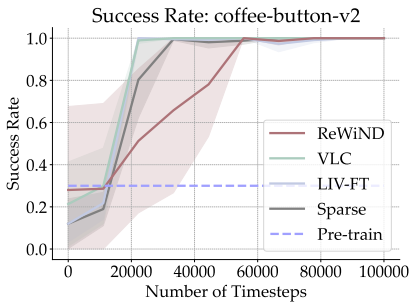



Fig. 12: **Metaworld Reward Confusion Matrix on 20 Training Tasks.** For each training task, we compute rewards for all combinations of demonstration videos and language descriptions. ReWiND produces the most diagonal-heavy confusion matrix, indicating strong alignment between unseen demos and instructions.

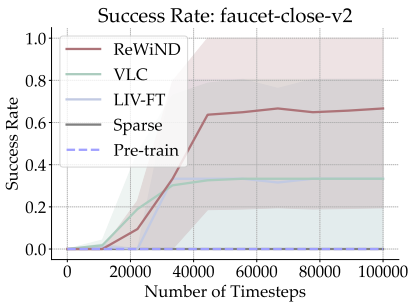
rewards regardless of input. To avoid overfitting, the main ReWiND model uses only first-frame positional embeddings (Section II-A2).



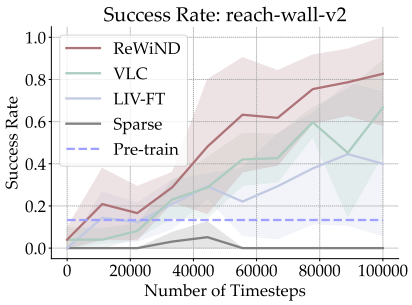

(a) Button Press (Wall)



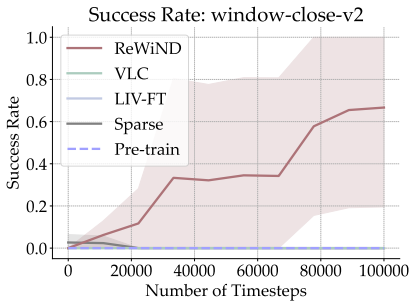

(b) Coffee Button



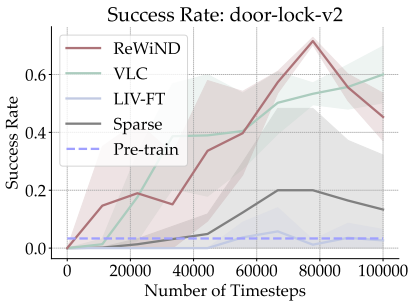

(c) Faucet Close



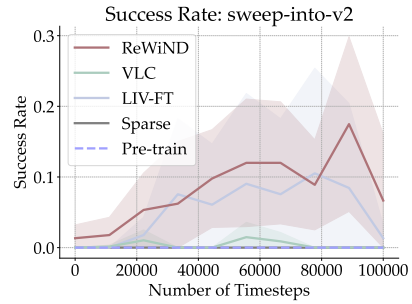

(d) Reach (Wall)



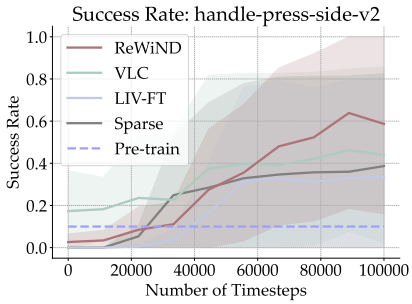

(e) Window Close



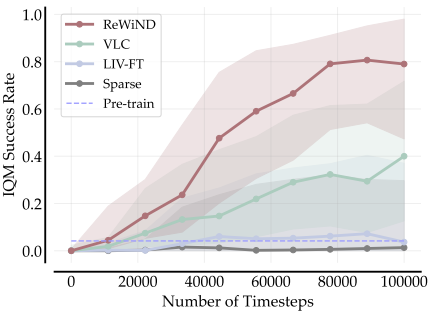

(f) Door Lock




(g) Sweep Into




(h) Handle Press (Side)




(i) All Tasks IQM and 95% CI

Fig. 13: **Metaworld success curves.** Task-level success rate learning curves plotting mean and shaded standard deviations. The bottom right figure plots the overall average across all tasks in terms of IQM and 95% confidence intervals.

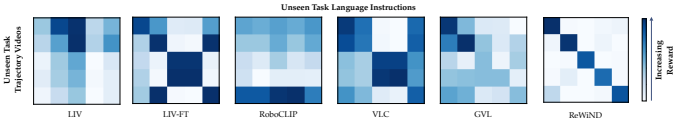



Fig. 14: **Real-world Koch Reward Confusion Matrix on 5 Unseen Tasks.** For each unseen task, we compute rewards for all combinations of demonstration videos and language descriptions. ReWiND produces the most diagonal-heavy confusion matrix, indicating strong alignment between unseen demos and instructions.

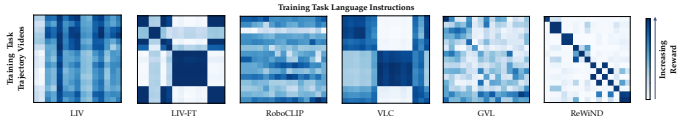



Fig. 15: **Real-world Koch Reward Confusion Matrix on 15 Training Tasks.** For each training task, we compute rewards for all combinations of demonstration videos and language descriptions. LIV-FT, VLC and ReWiND are pretrained or fine-tuned with these training task while LIV , GVL and RoboCLIP are not