# GenFlowRL: Shaping Rewards with Generative Object-Centric Flow in Visual Reinforcement Learning

Supplementary Material

## 1. Cross-Embodiment and Cross-Domain Data Collection

To train the flow generation model, we collect a dataset contains $12k$ trajectories of three different embodiments, where they work with ten different tasks from two different task domains. We aim to utilize different embodiments for highlighting that object-centric flow can be trained with large scale diverse training data.

In the setting of Im2Flow2Act [11], we utilize their sphere robot dataset as the first kind of cross-embodiment data. Four tasks, which are *PickNPlace*, *Pouring*, *Opening*, and *Folding*, are used for data collection. Also shown in Im2Flow2Act [11], this kind of data are proposed to emulate the cross-embodiment human data in the real world. Out of those tasks, we design a new contact-rich manipulation task *Pivoting* with the UR5 robot for collecting data. To collect robot data, we place the robot in different initial positions and use a manipulation script to move it to five different contact points. Then, we apply five different action scripts to enable the robot to stand the peg up and make contact with the wall. For data in the MetaWorld [12], the task setting and robot used for data collection was totally different from the Im2Flow2Act[11]. We choose five different tasks, which are *Assembly*, *Coffee Push*, *Door Close*, *Lever Pull*, and *Stick Pull*. Those data are collected by the Sawyer Robot. In this benchmark, we rollout the trained RL model in MetaWorld [12] for data collection. Each task contain 1200 trajectories, where we have 12k training data in total. The visualization of each embodiment are shown in Fig. 1.

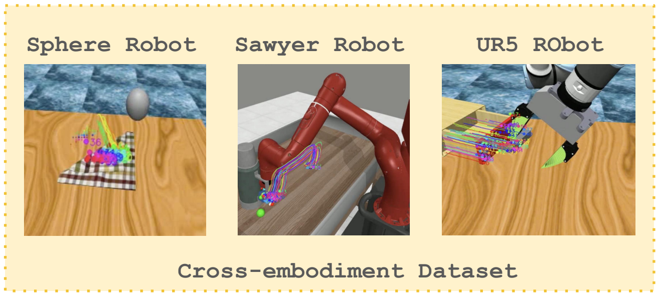

Figure 1. **Visualization of cross-embodiment data.**

## 2. Tasks Descriptions

We select four tasks from Im2Flow2Act [11], five tasks from MetaWorld [12], and one self-designed task *pivoting*.

- *PickNPlace:* Pick a mug from one random position to the bowl in another random position.
- *Pouring:* Pick a mug from one random position and pour the water to a bowl in another random position.
- *Opening:* Grasp the handle and open the cabinet.
- *Folding:* Fold the cloth from one corner to another corner.
- *Pivoting:* Contact with the peg without grasping, and pivot it to stand up by interacting with the wall.
- *Coffee Push:* Push the coffee mug to a specific position.
- *Door Close:* Close the door of a cabinet.
- *Assembly:* Pick up a stick and align its square hole with a peg on the table.
- *Lever Pull:* Grasp the lever and pull it up to the up right position.
- *Stick Pull:* Pick up a stick, insert it into a kettle, and pull the kettle to a specific position.

## 3. Flow Generation Implementation Details

In this section, we aim to share more details about implementation, training, and processing of our flow generation model, which is similar to Im2Flow2Act [11].

### 3.1. Implementations

The first step is getting the training dataset. We use Grounding DINO [7] to detect the bounding box of the described object from the initial RGB frame, and then uniformly sample keypoints within the box. To track those keypoints from the video, we apply the SOTA keypoint tracking foundation model [5] for keypoint tracking. Unlike the TAPIR [1] used in previous work [11], CoTracker [5] can track occluded objects in the image, which is extremely important for contact-rich manipulation tasks. Then, we formulate the tracked keypoints as object-centric flow $\mathcal{F}_0 \in \mathbb{R}^{3\times T\times H\times W}$ with temporal representation in T time space. The first two channels represent the pixel coordinates of object keypoints in image space, while the third represents their visibility during the execution.

The generated flow is conditioned on the initial image of the task, the initial keypoints, and the text description. The encoder deisgn for the inputs are also the same as the [11]. The text descriptions are processed into the CIIP [9] to obtain text embeddings. For the initial image, we utilize the CLIP encoder to get the patch embeddings. The initial keypoints are encoded through fixed 2D sinusoidal positional encoding. Finally, those inputs are processed into the denosing process through cross-attention.

With this flow representation, we can leverage the diffusion-based video generation based on AnimateDiff

[18] for flow generation. Same as the Im2FLow2Act [11], we encode the object flow into a latent space and train the generative model based on it. Similar to the StableDiffusion [10], we use the auto encoder VA-GAN [2] to encode the flow into low dimentional embeddings. Then, to utilize the low-dimensional latent space, we utilize a two-stage training process. Firstly, we fix the encoder from the AE and finetune the pretrained decoder to better adapt it to the flow images. Then, same as the Im2Flow2Act, we insert the motion module layer into StableDiffusion proposed by Animatediff [3] to model the temporal dynamics for flow generation. The second stage is training the motion module layer from scratch but only insert LoRA (Low-Rank Adaptation) layers [4] into the SD model.

### 3.2. Training Details

Training with cross-embodiment data from two different task domains, we process both the image from the tasks in Im2Flow2Act [11] and MetaWorld [12] into resolutions $480 \times 480$. For extracting keypoints from the bounding box generated by Grounding Dino [7], we set the spatial and temporal resolution to H = W = 32 and T = 32 for generating flow for 1024 keypoints over 100 steps, which also means that 100 keypoints set can be used for reward shaping in our reward model. To train the model on our cross-embodiment dataset, we firstly train the decoder of VQ-GAN [2] in StableDiffusion [10] for 400 epoches with a learning rate of $5e-5$. Secondly, for training AnimateDiff, we insert the LoRA [4] with a rank of 128 into the Unet from StableDiffusion and train the motion module layer from scratch with the same hyperparameter shown in Im2Flow2Act [11], which is trained with learning rate of $1e-4$ for 300 epochs using Adamw [8] optimizer with weight deacy $1e-2$ betas $(0.9, 0.999)$, and epsilon $1e-8$.

### 3.3. Flow Post Processing

Similar to Im2Flow2Act [11], we use motion filter to extract moving keypoints from the object itself. More specifically, we use moving filter to remove those static keypoints and use SAM [6] filter to remove keypoints which are not on the object, such as keypoints on the robot or the table.

**Moving Filter:** Since some of the keypoints selected from bounding box are sampled from the environment, we use the moving filter to extract the moving points from those keypoints. Then, we use moving filter to remove those keypoints whose movement in the image space ($480 \times 480$) is below a certain threshold. For all the tasks, we select 50 pixels as the threshold for removing those static keypoints. This method can effectively remove those background keypoints.

**SAM Filter [6]:** Since we also use robot data in our dataset, those keypoints on the robot are also will be counted as moving keypoints. Then, using SAM [6] to do semantic segmentation and remove those moving keypoints on the robot is necessary. We utilize SAM to obtain the segmentation and then iterate through the keypoints, filtering out those whose corresponding segment area exceeds a predefined threshold. To preserve keypoints on objects with rich textures, we set a high threshold of 10,000 across all tasks.

Finally, we randomly sampled 128 points from the selected keypoints for our reward model and policy input.

## 4. RL Implementation Details

### 4.1. Reward and Policy Input with Generated Flow

For both RL training in Im2Flow2Act [11] and MetaWorld [12], we firstly need to generate the initial flow from the first frame for each iteration, which will be used for both policy input and reward shaping. Then, the observation space of the policy input is $128 \times 3$, where we sampled 128 keypoints from the object.

To do online flow matching with the generated flow, we also utilize CoTracker [5] for online tracking in the real-robot execution. Using the same motion filters, 128 keypoints will be sampled from the realtime motion. The online keypoints tracking will be generated for each timestep during the robot execution. Then, those keypoints will be processed as $\delta$-flow for reward shaping.

Since our flow generation model will generate 100 frames with keypoints subset, we can use each centroid calculated from the keypoints subset of those 100 frames as reward. For reward calculation and policy input. We limit the $max\_step\_episode$ into number less or equal to 100 steps (respectively for different tasks) and use them for real-time flow matching in reward generation and policy learning.

### 4.2. RL Reward Design

Out of the flow-derived reward, our RL training Pipeline also requires specific reward design for achieving the goal.

**Reaching Reward:** For all the tasks, we need to define similar reaching reward to guide to robot move toward the object. For tasks which required robot to grasp the object, robot needs to open their gripper and reach the grasping position. For the tasks which required robot to push or contact with the object, the robot will be guided to move to contact with a certain area. The reward will be define as: $(1 - \tanh(10.0 \cdot d_{\text{grip}}))$.

**Grasping / Contact Reward:** We also design sparse reward as a subgoal for guiding robot to accomplish the task. For grasping task, we will set the reward to be $0.25$ as the reward. For contact reward, we will set the reward to be $0.25$ once the gripper is contact with a certain area of the object.

**Goal-conditioned Reward:** For all the task, we need to define a goal state to showcase that the robot successfully acchieve the goal. For most of the task, it should be easily, such as those defined task in MetaWorld [12] and some simple task like *PickNPlace*. We can just define the final position for object to be. For some other harder-to-define tasks like pouring, we set up a target pose range as the goal, which is limited to a certain position with certain orientations, where the orientation is sampled from $(\frac{5\pi}{16}\frac{7\pi}{16})$. For opening, the reward will be defined by the opening distance, which is $0.1m$.

### 4.3. Training Details

The Training hyperper parameters have been shown in Table 1.

Table 1. Hyperparameters for DrQv2 with Flow-derived Reward.

| **Hyperparameter** | **Value** |
|---|---|
| **Environment** | |
| Action repeat | 3 (MetaWorld) |
| | 3 (Im2Flow2Act) |
| Frame stack | 1 |
| Rendered Image | $480 \times 480$ |
| Observation size | $128 \times 3$ |
| Reward type | Sparse |
| **DrQv2** | |
| Data Augmentation | $\pm 4$ RandomShift |
| Replay buffer capacity | $10^6$ |
| Discount $\gamma$ | 0.99 |
| $n$-step returns | 3 |
| Seed frames | 4000 |
| Exploration steps | 2000 |
| Exploration stddev. clip | 0.3 |
| Exploration stddev. schedule | Linear(1.0, 0.1, $3 \times 10^6$) |
| Soft update rate | 0.01 |
| Optimizer | Adam |
| Batch size | 256 |
| Update frequency | 2 |
| Learning rate | $10^{-4}$ |

### 4.4. Visualization of delta-flow model

To formulate the delta-flow reward model, we first calculate the centroid of the flow at each time step and calculate the relative translation and rotations between the current step and initial step. More visualization of the calculated centroid at each time step is shown in Fig. 2.

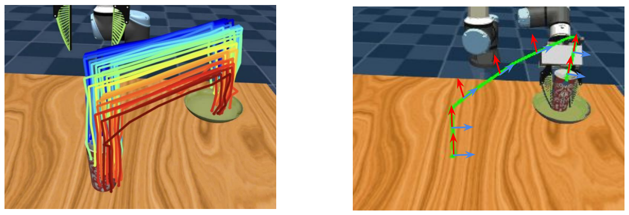
Figure 2. Visualization of the 2d delta-flow extractions.

## 5. Ablation Study

### 5.1. Ablation Study for Robustness of reward model

In real-world scenarios, the flow trajectory predicted by the diffusion model or the tracker can be noisy. To investigate the effect of these noises on policy performance, we simulate real-world noise for our method. Specifically, we build a noise model to cumulatively perturb trajectories and deviate endpoints, using the addition of Brownian motion (*i.e.*, Gaussian random walk) and Brownian bridge (*i.e.*, trajectories with predefined perturbed endpoints), with separate controllable standard deviations. We set up four types of random noise: small Gaussian (gauss=1x, drift=0x), large Gaussian (gauss=4x, drift=0x), small drift (gauss=2x, drift=1x), and large drift (gauss=2x, drift=2x). A vivid visualization of our noise composite model applied to a Bessel smoothed trajectory is displayed in Fig. 3.

We evaluate the performance of our model after applying this noise model to the generated flow in five challenging tasks. The result is shown in Table 2. Since our model is already trained on generated flows with different magnitudes of noise and due to our task-oriented rewards, it has the robustness to noise in the generated flows. Therefore, our method still achieves a similar performance with trajectory noise, especially for the cases with large Gaussian noises.

Furthermore, our method can maintain relatively high performance when the goal position is largely drifted from the ground-truth by $\geq$ 20 pixels in the tasks with position-sensitive evaluation, *e.g.*, `Folding` and `Pouring` (Fig. 3).

| | PickNP. | Pour | Open | Fold | Pivot |
|---|---|---|---|---|---|
| GENFLOWRL | 95 | 95 | 95 | 80 | 85 |
| +Gauss$\times$1 Drift$\times$0 | 95 | 95 | 90 | 80 | 85 |
| +Gauss$\times$4 Drift$\times$0 | 95 | 90 | 90 | 75 | 80 |
| +Gauss$\times$2 Drift$\times$1 | 95 | 90 | 90 | 70 | 85 |
| +Gauss$\times$2 Drift$\times$2 | 85 | 75 | 85 | 65 | 75 |

Table 2. Performance for noise sensitivity analysis on five tasks. Noises are added to flow trajectories for comparisons.

### 5.2. Ablation Study of Different Number of Keypoints

To conduct more evaluations with different number of keypoints, we evaluate the performance with 32 keypoints and 64 keypoints instead of 128 keypoints in the experiments. We find that the performance is similar to using 128 keypoints, which highlights that the stability of the delta-flow fomulations. The results is shown in Fig. 4.

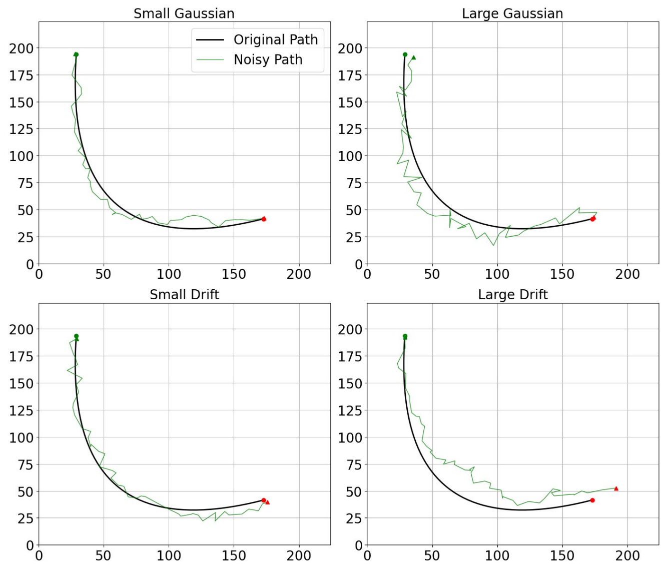

Figure 3. Visualization of the simulated noised 2D trajectory.

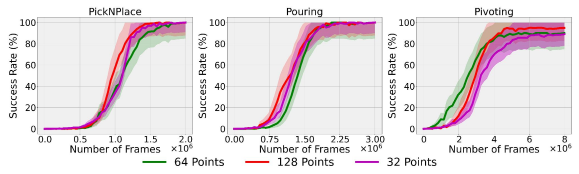

Figure 4. Results of RL performance with different number of keypoints.

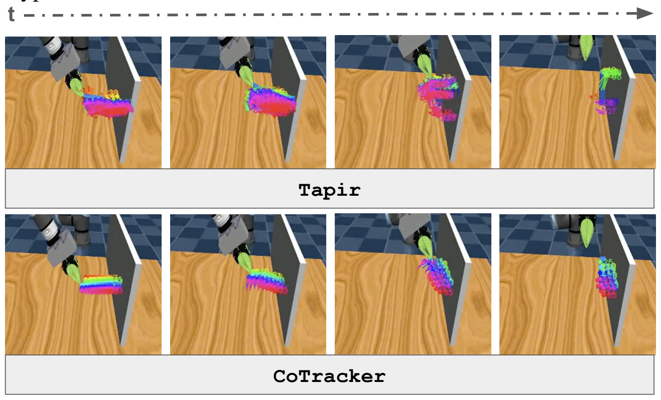

Figure 5. Visualization of the comparison of trackers.

### 5.3. Ablation Study of Tracking Model

Compared with Im2Flow2Act [11], we propose to use CoTracker [5] instead of [1] since Cotracker is better for occupation, which is important for contact-rich manipulation tasks.

To evaluate the performance of them, we visualize the qualitative results of them for the contact-rich manipulation task *Pivoting*, which is shown in Fig. 7. Through qualitative results, the CoTracker has much better performance than Tapir when the object meets obstruction.

## 6. Real World Case Study Implementation Details

In this section, we set collect actions from robot actions script with a limit for 200 steps for *Folding* and *PickNPlace*, and 100 steps for *Pouring* and *Pivoting* respectively. The robot control frequency and the CoTracker frequency are both 2.5 Hz. For collecting human demonstrations, the frequency of the CoTracker is 5 Hz, and the final number of steps are the same as the robot demo. Then, we calculate the flow matching reward between the human trajectory and robot trajectory.

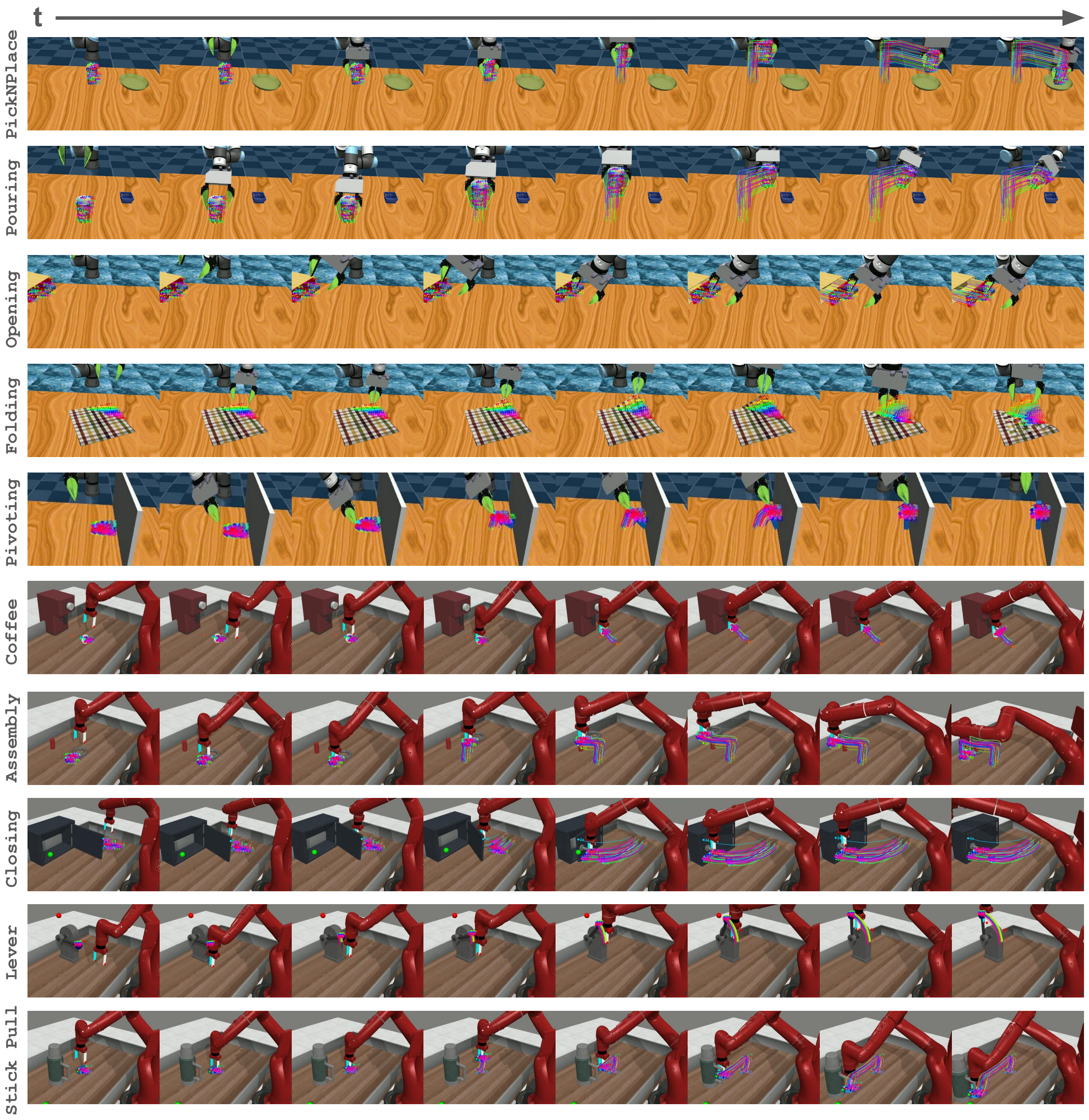

Figure 6. The qualitative result of the policy rollout in simulatior.

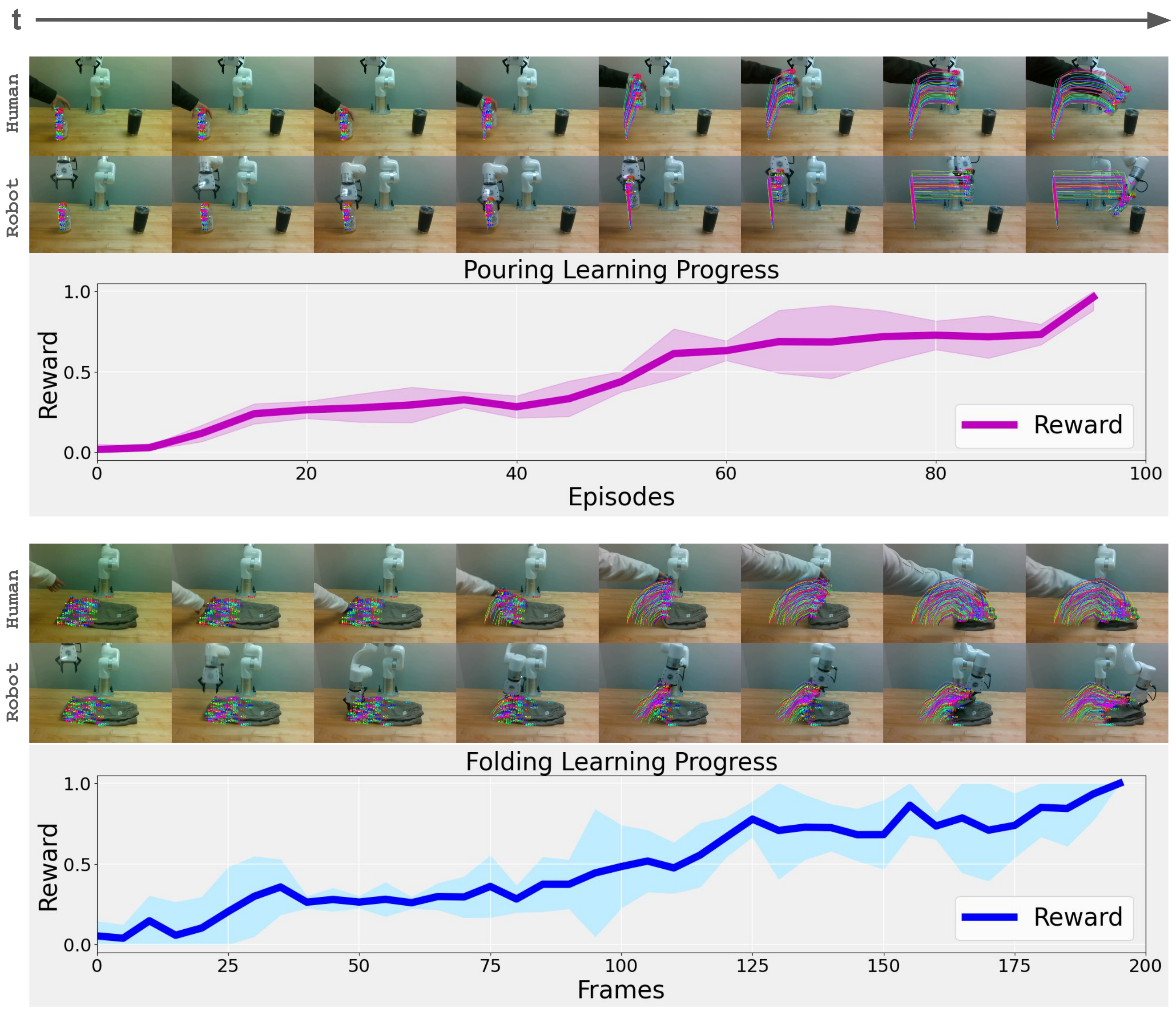

Figure 7. The qualitative result of the Flow Matching Reward Case Study in the real world.

## References

[1] Carl Doersch, Yi Yang, Mel Vecerik, Dilara Gokay, Ankush Gupta, Yusuf Aytar, Joao Carreira, and Andrew Zisserman. Tapir: Tracking any point with per-frame initialization and temporal refinement, 2023. 1, 4

[2] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. *CoRR*, abs/2012.09841, 2020. 2

[3] Yuwei Guo, Ceyuan Yang, Anyi Rao, Zhengyang Liang, Yaohui Wang, Yu Qiao, Maneesh Agrawala, Dahua Lin, and Bo Dai. Animatediff: Animate your personalized text-to-image diffusion models without specific tuning, 2024. 2

[4] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. 2

[5] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Co-Tracker: It is better to track together. 2023. 1, 2, 4

[6] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Paul Rolland, Laura Gustafon, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023. 2

[7] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *European Conference on Computer Vision*, pages 38–55. Springer, 2025. 1, 2

[8] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. 2

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. 1

[10] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. 2

[11] Mengda Xu, Zhenjia Xu, Yinghao Xu, Cheng Chi, Gordon Wetzstein, Manuela Veloso, and Shuran Song. Flow as the cross-domain manipulation interface, 2024. 1, 2, 4

[12] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021. 1, 2, 3