

## Supplemental Material

### A. Framework Details

Our model adopts a diffusion-based framework that incorporates cycle-aware perception and action generation capabilities. **Architecture overview:** The model takes multi-modal inputs including point clouds, proprioceptive states, and language instructions, encodes them through specialized modules, and fuses them into a unified conditional feature for action prediction via diffusion process. **Key components:** Cost-aware temporal sampling for historical perception, language-guided diffusion policy, multi-task learning for historical understanding.

#### A.1. Network Architecture

**Input modalities.** The model takes three types of inputs: single-view point clouds as high-overload observation  $o_i^h \in \mathbb{R}^{N \times 3}$ , language instructions  $lan$  describing the task and cycle count, and robot proprioception as low-overload observation  $o_i^l \in \mathbb{R}^M$  capturing joint states and gripper configurations.

**Cost-aware Sampling Strategy** To enable the model to better understand global information, we employ cost-aware sampler with different sampling strategy for different observation. For high-overload observation, as shown in Figure 1, we perform a boundary-biased binary partition over the interval [start, end), recursively selecting midpoints while ensuring inclusion of the segment’s earliest states for  $\frac{T}{2}$  frames. Additionally, to ensure action continuity, we sparsely sample  $\frac{T}{2}$  frames closer to the current time step using an exponential sampling strategy. For low-overload observation, we sample all past observation.

**Point Cloud Encoder.** We first crop the point cloud using a 3D bounding box following [13] and downsample it to  $N = 2048$  points using Farthest Point Sampling (FPS). We define the observation horizon as  $K_{high} = 6$  frames. Different from [13] that uses  $K_{high}$  consecutive frames in the past as input, we adopt a cost-aware sampling strategy to obtain  $K_{high}$  observations from longer perception horizon. We then employ a point encoder [13] to independently encode each frame, and finally concatenate the  $K_{high}$  features along the channel dimension to obtain the final representation  $f_h$ .

**State Encoder.** We encode proprioceptive trajectories using a downsampling attention encoder that first performs aggressive temporal reduction and then applies multi-head attention. Specifically, we employ a 1D convolution with stride 5 to extract action-chunking features by compressing the raw state sequence into compact tokens. These tokens, which are augmented with positional embeddings and concatenated by a learnable CLS token, are then fed into the Transformer layers. The final proprioceptive embedding is

taken from the CLS token as  $f_l$ .

**Language Encoder.** We use a CLIP [9] text encoder to extract language features. The extracted feature is then projected into  $F_l = 160$  dimensions via a linear layer as language feature  $f_{lan}$ .

**Feature Fusion and Conditioning.** We employ a multi-layer MLPs  $\phi_{dense}$  to fuse  $f_h$  and  $f_l$  into  $f_{lh}$ , which is then concatenated with  $f_{lan}$  to form the final conditional feature  $f_c$ . We employ FiLM [7] conditioning to inject the conditional feature  $f_c$  into the denoising network. Specifically, the conditional feature modulates the intermediate layers of the denoising U-Net through affine transformations, enabling the network to adapt its behavior based on the multi-modal observations and task instructions.

**Denoising Network and Action Generation.** The denoising network takes as input the noisy action  $\mathbf{a}_t$  at diffusion timestep  $t$  and the conditional feature  $f_c$ , and predicts the noise  $\epsilon$  that was added to the clean action  $\mathbf{a}_0$ . The network architecture is based on a 1D temporal U-Net with multiple residual blocks. Each block receives the FiLM-modulated [7] features to condition the denoising process on the current observation and task. During training, we sample actions from expert demonstrations and apply forward diffusion to corrupt them with noise. During inference, we start from pure Gaussian noise and iteratively denoise it through the reverse diffusion process to generate the final action sequence  $\mathbf{a} \in \mathbb{R}^{k_a}$ , where  $k_a$  denotes the action dimension specific to the task (e.g., joint positions, velocities, and gripper states).

#### A.2. Ground Truth Generation

We employ a multi-task learning strategy with two supervision signals: **action ground truth** for the diffusion model and **progress ground truth** for the auxiliary progress prediction task.

**Action Ground Truth.** For the diffusion model, the ground truth is the clean robot action sequence  $\mathbf{a}_0 \in \mathbb{R}^{k_a}$  extracted from expert demonstrations, where  $k_a$  denotes the action dimension specific to the task. During training, we apply the forward diffusion process to corrupt the clean action with Gaussian noise:

$$\mathbf{a}_t = \sqrt{\bar{\alpha}_t} \mathbf{a}_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, \quad (1)$$

where  $\epsilon \sim \mathcal{N}(0, \mathbf{I})$  is the Gaussian noise,  $t$  is the diffusion timestep, and  $\bar{\alpha}_t$  is the cumulative noise schedule coefficient. The network  $\epsilon_\theta$  is trained to predict the added noise  $\epsilon$  given the noisy action  $\mathbf{a}_t$  and the timestep  $t$ .

**Progress Ground Truth.** To enable the model to better understand the history context and task progress, we introduce an auxiliary progress prediction task. The ground truth

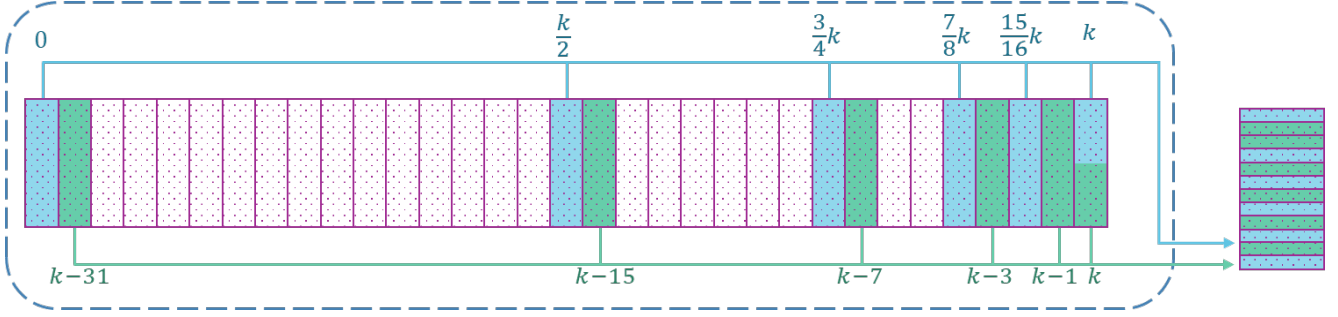


Figure 1. The visualization of boundary-biased binary partitioning. We perform a boundary-biased binary partition sampling and exponential sampling.

progress  $b_t$  is computed as:

$$b_t = \frac{t_{\text{current}}}{t_{\text{max}}}, \quad b_t \in [0, 1], \quad (2)$$

where  $t_{\text{current}}$  is the current frame number and  $t_{\text{max}}$  is the maximum frame number for the task trajectory. Then we uniformly partition the interval  $[0, 1]$  into ten bins and discretize  $b_t$  into the corresponding class label  $y_t$ , which is then used for a 10-way classification objective.

$$y_t = \lfloor 10 b_t \rfloor, \quad y_t \in \{0, 1, \dots, 9\}. \quad (3)$$

**Progress Prediction Head.** The progress prediction head takes the fused feature  $f_{lh}$  and processes it through a lightweight single layer MLP network  $\phi_{\text{light}}$ :

$$\hat{b}_t = \phi_{\text{light}}(f_{lh}), \quad (4)$$

where the single-layer MLP  $\phi_{\text{progress}}$  is intentionally kept lightweight to mitigate the risk of overfitting the multi-task head.

### A.3. Loss Functions

**Diffusion Loss.** The primary training objective minimizes the denoising score matching loss, formulated as:

$$\mathcal{L}_{\text{diffusion}} = \mathbb{E}_{\mathbf{a}_0, \epsilon \sim \mathcal{N}(0, \mathbf{I})} \left[ \|\epsilon - \epsilon_{\theta}(\mathbf{a}_t, t)\|^2 \right], \quad (5)$$

where the network learns to predict the noise added to the clean action at each diffusion timestep.

**Progress Classification Loss.** The auxiliary progress loss is defined as a cross-entropy loss:

$$\mathcal{L}_{\text{progress}} = \mathbb{E}_{b_t} \left[ - \sum_{k=1}^{10} b_t^{(k)} \log \hat{b}_t^{(k)} \right], \quad (6)$$

where  $b_t^{(k)}$  denotes the one-hot ground-truth label and  $\hat{b}_t^{(k)}$  is the predicted probability for the  $k$ -th cycle stage.

**Total Training Objective.** The total training objective combines both losses:

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_{\text{diffusion}} + \beta \mathcal{L}_{\text{progress}}, \quad (7)$$

where  $\alpha$  and  $\beta$  are weighting coefficients that balance the contributions of the diffusion loss and the progress prediction loss, respectively.

**Inference Sampling.** We employ DDIM [10] for efficient inference sampling. The reverse diffusion process is formulated as:

$$\begin{aligned} \mathbf{a}_{t-1} = & \sqrt{\bar{\alpha}_{t-1}} \left( \frac{\mathbf{a}_t - \sqrt{1 - \bar{\alpha}_t} \epsilon_{\theta}(\mathbf{a}_t, t)}{\sqrt{\bar{\alpha}_t}} \right) \\ & + \sqrt{1 - \bar{\alpha}_{t-1}} \epsilon_{\theta}(\mathbf{a}_t, t), \end{aligned} \quad (8)$$

where  $\bar{\alpha}_{t-1}$  and  $\bar{\alpha}_t$  are the cumulative noise schedule coefficients at time steps  $t-1$  and  $t$ , respectively.

## B. Benchmark Details

We construct comprehensive benchmarks for cyclic manipulation in both simulation and real-world settings. **The simulation benchmark** comprises eight diverse cyclic tasks with 200 demonstrations per task (1600 total), built on the SAPIEN [12] physics simulator with automatic data generation pipeline. **The real-world benchmark** includes six representative tasks across multiple robot embodiments, like bi-arms grippers, dexterous hands and humanoids.

### B.1. Simulation Benchmark

#### B.1.1. Overall

Our simulation benchmark is built upon the RoboTwin 2.0 platform [1], utilizing the SAPIEN physics simulator for high-fidelity robotic manipulation. **Dataset composition:** We collect 200 expert demonstration trajectories for each of eight tasks, systematically covering loop counts from 1 to 8 cycles with 25 trajectories evenly sampled per cycle count. All demonstrations are recorded at 30Hz control and observation frequency. **Multi-modal data:** Each trajectory cap-

tures single-view point clouds (2048 points via FPS down-sampling), RGB-D images from head-mounted camera (Intel RealSense D435), and proprioceptive state comprising joint positions, velocities, and gripper states. **Robot platforms:** The chemical mixing task uses an ALOHA-AgileX setup, whereas the remaining tasks use two ARX-X5 arms positioned 0.6 m apart. **Annotations:** Every trajectory is annotated with the target cycle count specified in natural language instructions, the completed cycle count at each timestep (providing frame-level progress supervision), 6D poses of key task-relevant objects tracked throughout execution, and complete arm joint configurations and end-effector poses for both arms. Note that the Morse tapping task follows a fixed pattern ("SOS") rather than varying cycle counts.

### B.1.2. Environment Setup

The environment setup involves several key components for automatic and scalable data generation:

**3D Assets.** We create a diverse set of 3D assets by leveraging existing object datasets from RoboTwin [1] and generating textured 3D models from single-view RGB images using Hyper3D [3]. We manually adjust the scale of generated models to ensure suitability for robotic manipulation. Each manipulable object is annotated with *contact points* that define task-relevant interaction locations and orientations, stored in JSON format with both position  $(x, y, z)$  and orientation (quaternion) information.

**Automatic Grasping System.** The framework provides an automatic grasping system that generates collision-free manipulation trajectories by motion planner. For each manipulation primitive, the system enumerates all annotated functional points, evaluates candidate grasps by computing quaternion distances to preferred orientations (weighted combination:  $0.7 \cdot d_{\text{top-down}} + 0.3 \cdot d_{\text{side}}$ ), verifies kinematic feasibility, and selects the optimal configuration. The system then generates a three-step action sequence: moving to a pre-grasp pose (0.1m approach distance) via collision-free RRT planning [4], approaching the grasp pose with constrained motion, and closing the gripper to the specified width. Planned trajectories are time-parameterized using TOPPRA [8] to compute velocity profiles  $s(t)$  that respect joint velocity and acceleration limits while maintaining 250Hz control frequency, with dual-arm coordination achieved through proportional progress synchronization.

**Automatic Cycle-data Generation.** Our data collection system integrates sophisticated loop control functionality through a configurable parameter system. For each task, we define a `play_once(loop_times)` method that encapsulates a complete episode with the specified number of cycles as input. This method internally maintains loop counters that track the progress of cyclic actions, executing repetitive motion primitives (e.g., shaking, hammering, cutting) for the exact number of times specified. The pa-

rameterized loop times range from 1 to 8 cycles per episode, providing comprehensive coverage of different task lengths. The system supports randomization of initial object poses within predefined workspace bounds, ensuring natural variations in starting configurations.

**Language Instruction Generation.** We leverage a template-based approach that incorporates explicit cycle count information. We design instruction templates with a `[num]` placeholder to denote the number of repetitions required (e.g., "shake the bottle `[num]` times" or "hammer the red block `[num]` times"). During data generation, the system automatically replaces the placeholder with the actual cycle count, ensuring that language instructions explicitly convey the temporal extent of cyclic actions.

### B.1.3. Task Descriptions

**Block Hammering.** This task simulates using a hammer to strike objects. The robot grasps a hammer and uses it to strike a stationary block repeatedly. Each cycle consists of lifting the hammer and placing it down onto the block's functional point.

**Bottle Shaking.** This task simulates mixing liquid by shaking a bottle. The robot grasps a bottle placed randomly on the table and performs shaking motions by alternating between two orientations.

**Carrot Cutting.** This task simulates cutting vegetables with a knife. The robot grasps a knife and performs repetitive cutting motions on a stationary carrot. Each cycle involves lowering the knife onto the carrot and lifting it back up. The knife must traverse leftward incrementally to simulate progressive cutting.

**Dual-knife Chopping.** This bimanual task simulates using two hands to chop ingredients simultaneously. Two knives are positioned on opposite sides of the table. Both arms grasp their respective knives and perform synchronized chopping motions on a cutting board placed centrally. Each cycle consists of simultaneous downward chopping by both knives followed by lifting.

**Roller Rolling.** This bimanual task simulates rolling out dough with a rolling stick. The robot grasps a rolling pin at two contact points and performs horizontal rolling motions on the table surface. Each cycle consists of a forward push followed by a backward pull.

**Morse Tapping.** This task simulates tapping a bell to transmit Morse code messages. The robot taps a bell to transmit the Morse code for "SOS" ( $\cdots - - - \cdots$ ). The tapping pattern includes short taps and long taps with different durations.

**Egg Beating.** This bimanual task simulates beating eggs in a bowl. One arm holds a bowl while the other arm operates an egg beater in circular stirring motions. Each cycle consists of a complete circular trajectory.

**Chemical Mixing.** This bimanual task simulates mixing chemical reagents by shaking a flask. One arm holds a flask

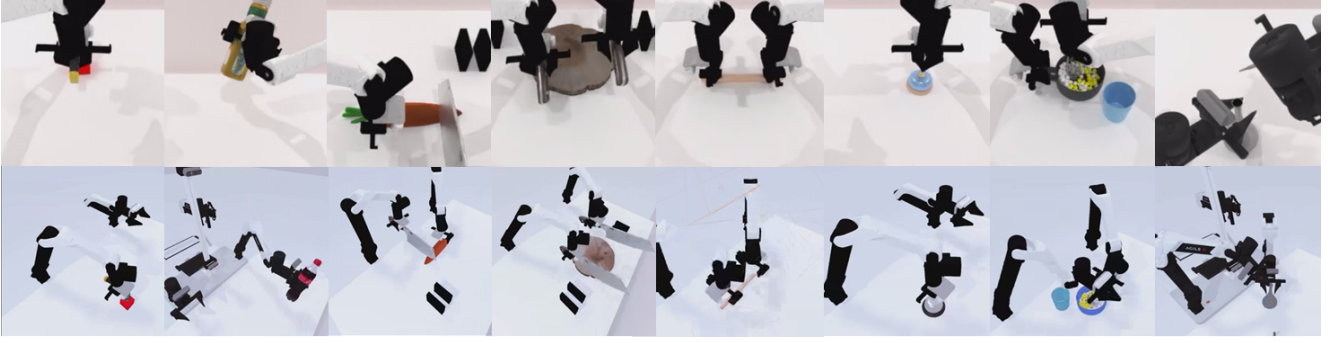


Figure 2. **Visualizations of the eight cyclic manipulation tasks in the simulation benchmark.** From left to right: Block Hammering, Bottle Shaking, Carrot Cutting, Dual-knife Chopping, Roller Rolling, Morse Tapping, Egg Beating, Chemical Mixing.

while the other arm operates a dropper positioned above the flask opening. The flask is shaken by alternating between two tilted orientations.

#### B.1.4. Automatic Evaluation

We design a comprehensive automatic evaluation system that combines multiple detection strategies to ensure accuracy and robustness.

**Contact-based Detection.** For tasks involving physical contact (hammering, cutting, Morse tapping), we employ a state-machine-based collision detection system that maintains a binary contact state and transitions between states only after observing consistent signals for  $N$  consecutive frames (where  $N$  is the state threshold: 2 frames for hammering, 5 frames for cutting), effectively filtering transient collision signals caused by simulation noise. Each transition from non-contact to contact state (False  $\rightarrow$  True) increments the cycle counter. The system records frame numbers where cycles occur  $\{f_1, f_2, \dots, f_k\}$ , inter-cycle gaps  $\{f_2 - f_1, f_3 - f_2, \dots, f_k - f_{k-1}\}$ , and contact state history for post-hoc visualization. Physical contact is detected via SAPIEN’s collision callback system.

**Peak Detection.** For non-contact cyclic motions (shaking, rolling, beating, mixing), we employ peak detection on object pose trajectories by extracting task-specific time-series signals from tracked object poses (e.g., z-axis orientation component for bottle shaking, y-position for roller rolling, angular position in cylindrical coordinates for egg beating). The peak detection algorithm first smooths the signal using a 15-frame sliding window, then applies `scipy.signal.find_peaks` with task-specific parameters including minimum peak height (20% of signal range), minimum separation between peaks (30 frames), and required prominence (4% of signal range). The system generates diagnostic plots showing the smoothed signal, detected peaks, and their positions for manual inspection.

**Hybrid Verification.** To enhance robustness, we implement a hybrid verification system for contact-based tasks that runs both collision detection and peak detection in par-

allel, assigning high confidence when results agree and logging supplementary notes when they differ, with the final reported cycle count using the collision-based result as primary and peak detection as validation. The visualization of detection methods is shown in Figure 3.

**Success Criteria.** A trajectory is marked successful if the detected cycle count matches the target cycle count, no catastrophic failures occur (object falling, robot collision, joint limits exceeded), and task-specific constraints are satisfied (e.g., bottle remains above table for shaking). The evaluation system generates detailed reports including total cycles completed, frame numbers of each cycle event, inter-cycle timing statistics (mean and standard deviation of gaps), and visualizations such as trajectory plots and contact state diagrams.

## B.2. Real-world Benchmark

### B.2.1. Overall

To validate the real-world applicability of our method, we construct a real-world benchmark spanning six representative cyclic manipulation tasks across multiple robot embodiments. **Robot platforms:** We use AgileX Piper for single-arm and dual-arm tasks, BrainCO Revo2 dexterous hands for bimanual dexterous tasks, and Unitree G1 humanoid robot for whole-body cyclic tasks. **Data collection:** Demonstrations are collected via teleoperation using Gello [11] for arm control, TypeTele [5] for dexterous hand control, and OpenWBC [6] for humanoid control. The dataset comprises demonstrations with varying cycle counts and control frequencies ranging from 10Hz to 50Hz depending on task requirements. Table 1 summarizes the data statistics for each task.

### B.2.2. Hardware Setup

**Robot Embodiments.** We use AgileX Piper for single-arm and dual-arm tasks, leveraging its 6-DoF arms and parallel grippers for versatile manipulation. Dexterous bimanual tasks are carried out on the BrainCO Revo2 dexterous



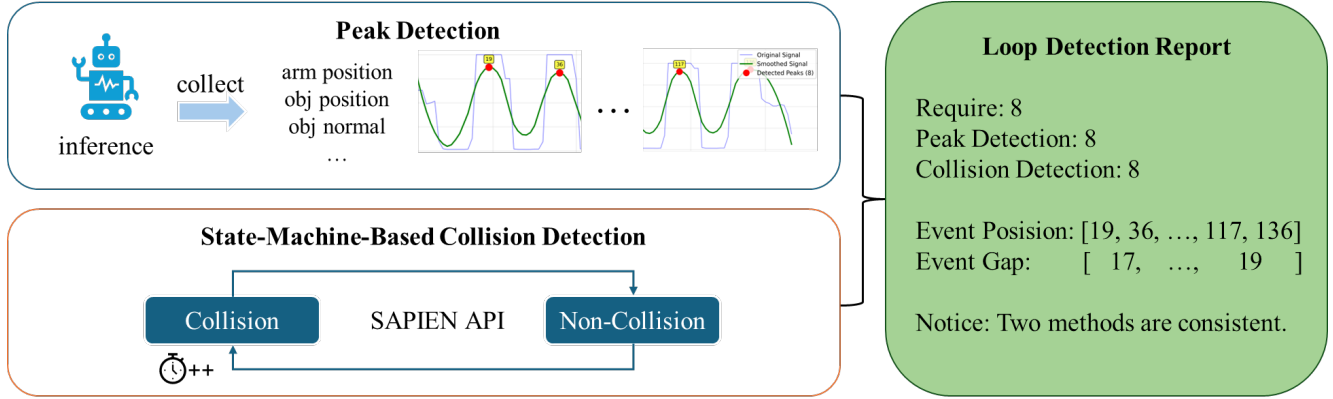


Figure 3. **Visualization of cycle detection methods.** (a) Peak detection on all eight tasks, showing smoothed signals with detected peaks marked. (b) State-machine-based contact detection for tasks involving physical contact. (c) Hybrid loop detection report combining both methods for enhanced robustness.

hands. Additionally, we employ the Unitree G1 humanoid robot for cyclic tasks requiring whole-body coordination.

**Visual Perception.** We mount an Intel RealSense L515 RGB-D camera on the third perspective (front view) of the Piper robot to capture high-fidelity point clouds of the workspace. We use 640x480 resolution for both RGB and depth images, aligning depth to color frames for accurate 3D reconstruction.

### B.2.3. Data Collection

**Teleoperation for Arm Control.** We use Gello hardware and software [11] to collect real-world demonstration data. Specifically, we build two isomorphic gello arms using Dynamixel motors and 3D-printed components. Each master arm has 6 degrees of freedom (DoF) and is equipped with a parallel gripper. In control loop, the master arm captures joint positions and velocities at 60 Hz, which are then transmitted to the corresponding slave arm (Piper robot) via CAN bus communication. The slave arm executes the received commands in position control mode using pipersdk.

**Teleoperation for Dexterous Hand Control.** For dexterous bimanual tasks, we use TypeTele [5] to teleoperate the BrainCO Revo2 dexterous hands. Specifically, we design types for each task and map the master gello gripper’s 0-1 opening range to the Revo2’s open/close type.

**Teleoperation for Humanoid Control.** For humanoid tasks, we use OpenWBC [6] to teleoperate the Unitree G1 humanoid robot.

### B.2.4. Task Descriptions

**Bottle Shaking.** Using single arm, the robot is required to grasp a bottle placed on the table and perform shaking motions for a specified number of cycles.

**Block Hammering.** Using single arm, the robot must grasp a hammer and strike a block repeatedly for a given number of cycles.

Task Name	Data Amount	Cycle Range	Frequency (Hz)
Bottle Shaking	150	1-8	20
Block Hammering	100	1-8	20
Carrot Cutting	50	1-8	10
Drum Beating	50	2-10	20
Table Cleaning	50	1-5	20
Tire Pumping	20	1-5	50

Table 1. Real-world data collection statistics for each task.

**Carrot Cutting.** Using both arms, the robot needs to grasp a knife and hold a carrot on the cutting board, performing cutting motions for a specified number of cycles.

**Drum Beating.** Using both arms, the robot is required to hold drumsticks tightly and beat the drum for a given number of cycles.

**Table Cleaning.** Using both arms and dexterous hands, the robot needs to grasp a trash shovel and a broom, and perform cleaning motions on the table surface for a specified number of cycles.

**Tire Pumping.** Using whole body, the robot is required to grasp a tire pump and perform pumping motions by squatting down and standing up for a given number of cycles.

### B.2.5. Evaluation Pipeline

**Evaluation Protocol.** We deploy the trained model on the real robot to perform each task, keeping the initial environment configuration the same. We strictly control the inference frequency to be consistent with the training data to ensure stable performance. For each task, we test twice for every cycle count in the training data range. For example, if the training data covers cycle counts from 1 to 8, we test each cycle count (1 through 8) twice, resulting in a total of

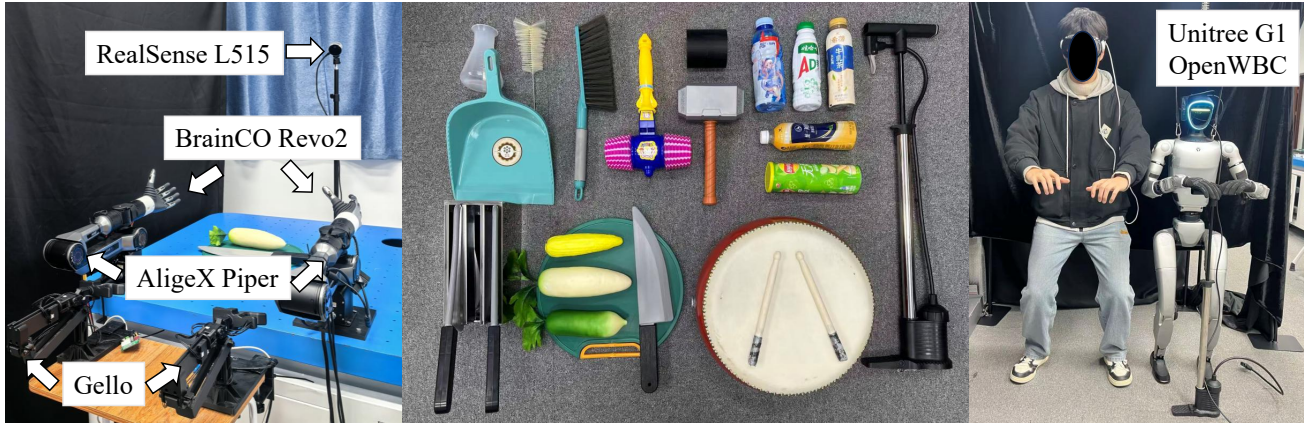


Figure 4. **The hardware setup for real-world benchmark.** (a) AgileX Piper robot (gripper and BrainCO Revo2 dexterous hands) used for single-arm and dual-arm tasks, equipped with an Intel RealSense L515 RGB-D camera for visual perception. (b) Object sets for real-world tasks. (c) Unitree G1 humanoid robot utilized for whole-body cyclic tasks.

16 evaluation runs per task.

**Success Criteria.** During each run, two or more observers independently record the number of successfully completed cycles to ensure accurate assessment. A run is considered successful only if all observers agree that the robot has completed the exact number of required cycles without any failures (e.g., dropping objects, incorrect motions). The final success rate for each task is computed as the ratio of successful runs to total runs conducted.

## C. Experiments Details

### C.1. Implement Details

**Training hyperparameters.** To ensure fair comparisons, we train all models (both baseline and our method) using the same training protocol and hyperparameters. We use the AdamW optimizer with learning rate  $1 \times 10^{-4}$ , betas  $[0.95, 0.999]$ , weight decay  $1 \times 10^{-6}$ , and batch size 128. Training proceeds for 600 epochs with a cosine learning rate schedule and 500 warmup steps. We employ Exponential Moving Average (EMA) with momentum 0.9999 on model parameters, activated after the first training step.

**Inference hyperparameters.** The noise scheduler follows a squared cosine schedule with 100 diffusion timesteps during training, with  $\beta_{\text{start}} = 0.0001$  and  $\beta_{\text{end}} = 0.02$ . For inference, we use DDIM sampling with 10 denoising steps to efficiently generate action sequences.

**Evaluation protocol.** In evaluation on the CycleManip benchmark, we assess model performance on all eight tasks, measuring success rates based on the automatic evaluation metrics described in Section 2.1.4. Each model is tested for 100 times per task to ensure statistical significance.

### C.2. Real World Deploy Details

**Training process.** We use the collected real-world demonstration data to train our CycleManip model for real-world deployment. The training process follows the same hyperparameters as in simulation, as described in Section 3.1.

**Deployment protocol.** In evaluation, we deploy the trained model on the real robot to perform each task, keeping the initial environment configuration consistent across runs. We strictly control the inference frequency to match the training data frequency (as shown in Table 1) to ensure stable performance.

**Evaluation setup.** For each task, we test twice for every cycle count in the training data range. For example, if the training data covers cycle counts from 1 to 8, we test each cycle count (1 through 8) twice, resulting in a total of 16 evaluation runs per task. The evaluation protocol and success criteria follow the metrics described in Section 2.2.5.

### C.3. Baseline Reproduce

**Traditional visuomotor policies.** When comparing with traditional visuomotor policies like dp3 [13], we add a language instruction encoder to process the cycle-aware language instructions and fuse it with other features as condition, while keeping all other components and hyperparameters identical to ensure a fair comparison. The language encoder architecture and fusion mechanism are the same as used in our CycleManip model.

**Vision-language-action models.** When comparing with vision-language-action models like Pi-0 [2], we follow the official training and inference protocols provided in their codebase. We use the same training data and evaluation metrics to ensure fair comparison.

454 **C.4. Details of General Manipulation Task**

455 To assess the general effectiveness of our method beyond  
456 cyclic tasks, we evaluate it on seven representative tasks  
457 from the RoboTwin benchmark [1]: Place-Cans-PlasticBox,  
458 Handover-Block, Pick-Diverse-Bottles, Stamp-Seal, Place-  
459 Bread-Basket, Open-Microwave, and Turn-Switch.

460 **Data collection.** We adopt the official `demo_clean`  
461 configuration to collect 50 demonstrations per task. Each  
462 demonstration is collected using the automatic data genera-  
463 tion pipeline described in Section 2.1.2.

464 **Training and evaluation.** We train a separate CycleMa-  
465 nip model for each task using the same hyperparameters as  
466 described in Section 3.1. Training and evaluation strictly  
467 follow the original RoboTwin protocol to ensure fair and  
468 reproducible comparisons. Each model is evaluated for 100  
469 episodes per task, and success rates are computed based on  
470 the official RoboTwin evaluation metrics.

471 **C.5. Details of Plug & Play**

472 To demonstrate the Plug-and-Play capability of our method  
473 in recent vision-language-action models, we integrate our  
474 core CycleManip module into the Pi-0 [2] framework.

475 **Integration method.** Specifically, we replace Pi-0’s  
476 original state perception module with our multi-modal fu-  
477 sion module (point cloud encoder, state encoder, and lan-  
478 guage encoder as described in Section 1.1), while keeping  
479 all other components unchanged including the action de-  
480 coder, loss functions, and training procedure.

481 **Evaluation protocol.** We evaluate the modified Pi-0 on  
482 the CycleManip benchmark to assess performance improve-  
483 ments brought by our cycle-aware perception and action  
484 generation capabilities. The evaluation follows the same  
485 protocol as described in Section 3.1, testing on all eight  
486 tasks with 100 episodes per task.

487 **C.6. Details of Computation Efficiency**

488 To evaluate the computational cost of our CycleManip  
489 model, we conduct comprehensive efficiency analysis on a  
490 single NVIDIA RTX 4090 GPU.

491 **Experimental setup.** We compare our method with the  
492 baseline [13] on the carrot cutting task in our benchmark.  
493 Both models are trained and evaluated using identical hard-  
494 ware and software configurations.

495 **Metrics.** We report the following metrics:  $\text{Time}_{\text{train}}$  (per-  
496 step training time in seconds),  $\text{Time}_{\text{test}}$  (inference time with  
497 10 diffusion steps in seconds),  $\text{GPU}_{\text{train}}$  (GPU memory us-  
498 age during training in GB), and  $\text{GPU}_{\text{test}}$  (GPU memory us-  
499 age during inference in GB).

500 **Results summary.** Our method achieves a higher suc-  
501 cess rate while incurring only marginally increased training  
502 and inference time, as well as slightly higher GPU memory  
503 consumption, demonstrating the efficiency of our approach.

**D. More Visualization**

We provide more visualization of autonomous inference of  
our framework in real world, as shown in Figure 5. To re-  
duce space usage, we omit the loop frames and display only  
a single loop action. This demonstrates that our autonomous  
framework can perform various cyclic tasks across different  
robots.

504

505  
506  
507  
508  
509  
510





Figure 5. Visualization of autonomous inference of our framework in real world. To reduce space usage, we omit the loop frames and display only a single loop action.



## References

- [1] Robotwin 2.0 benchmark leaderboard. <https://robotwin-platform.github.io/leaderboard>, 2025. 2, 3, 7
- [2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky.  $\pi_0$ : A vision-language-action flow model for general robot control, 2024. 6, 7
- [3] Hyper3D. Hyper3d: Ai-powered 3d model generator, 2024. 3
- [4] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998. 3
- [5] Yuhao Lin, Yi-Lin Wei, Haoran Liao, Mu Lin, Chengyi Xing, Hao Li, Dandan Zhang, Mark Cutkosky, and Wei-Shi Zheng. Typetele: Releasing dexterity in teleoperation by dexterous manipulation types, 2025. 4, 5
- [6] Jiacheng Liu. Openwbc: Vr-based robot teleoperation and data collection system for unitree g1. <https://github.com/jiachengliliu3/OpenWBC>, 2025. GitHub repository. 4, 5
- [7] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023. 1
- [8] Hung Pham and Quang-Cuong Pham. A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Transactions on Robotics*, 34(3):645–659, 2018. 3
- [9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021. 1
- [10] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 2
- [11] Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. Gello: A general, low-cost, and intuitive teleoperation framework for robot manipulators. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12156–12163. IEEE, 2024. 4, 5
- [12] Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11097–11107, 2020. 2
- [13] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. *arXiv preprint arXiv:2403.03954*, 2024. 1, 6, 7