

# SafeBimanual Supplementary

## A Video Demo

A short video shows **SafeBimanual** imposing safety constraints to a diffusion-based policy for bi-manual manipulation. The clip explains the idea, shows each step of the method, and ends with the long-horizon *Prepare Breakfast* task completed safely. Enjoy!

## B Task Description

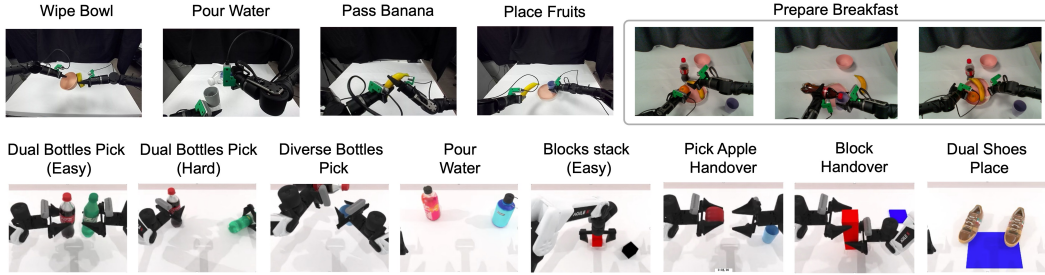


Figure 1: Snapshot of all simulation(second line) and real-world tasks(first line).

### B.1 Simulated Tasks

**Dual Bottles Pick (Easy):** Two upright bottles are randomly placed on the left and right sides of the table. Each arm must grasp its assigned bottle, lift it, and transport both to a shared target area. Success means both bottles end up upright and stably positioned within the target zone without dropping.

**Dual Bottles Pick (Hard):** Two bottles are randomly oriented (upright, tilted, or lying down) and placed left and right. Each arm must identify, grasp, and lift its bottle despite the varied poses, then place both upright in the target area. Success requires both bottles to be upright and stably located in the goal region.

**Block Handover:** A long rectangular block lies on the left side of the table. The left arm must pick it up, transfer it to the right arm, and the right arm must place it at a marked goal on the right. Success is achieved when the block is handed over and placed stably at the goal.

**Blocks Stack (Easy):** One red and one black cube are scattered randomly. The robot must stack the black cube on top of the red cube in the correct order. Success is defined by a stable vertical stack in the specified color sequence.

**Pour Water:** A filled bottle sits on the left and an empty cup on the right. The left arm must grasp and tilt the bottle over the cup held by the right arm to pour water. Success requires water to transfer into the cup without spilling outside.

**Pick Apple Handover:** An apple is placed at a random location on the left side of the table. The left arm must grasp the apple, transfer it to the right arm, and fully open its gripper to complete the handover. Success is achieved when the apple is securely held by the right arm and the left gripper is fully open with no residual contact.

**Dual Shoes Place:** Two shoes from the same pair, each with a distinct design, are placed randomly on the left and right sides of the table. The robot’s left and right arms must each grasp their assigned shoe and deposit it into the blue target area, ensuring the toe of each shoe points toward the table’s left or right edge.

**Diverse Bottles Pick:** Two bottles of different shapes and sizes are randomly positioned. Each arm must pick its assigned bottle and deliver both to a common target region. Success means both bottles are placed upright and stably in the goal area.

## B.2 Real-World Tasks

**Pass Banana:** A banana lies on the left side of the workspace. The left arm must grasp the banana, pass it to the right arm, which places it into the other side of table. Success is achieved when the banana is transferred intact and stably positioned within the target region.

**Pour Water (Real):** A filled bottle and an empty cup are arranged on a table. The robot must coordinate both arms to pour water from the bottle into the cup. Success requires sufficient water transfer with minimal spillage.

**Wipe Bowl:** A bowl sits on the left and a sponge on the right. The left arm must lift and stabilize the bowl while the right arm wipes its interior with the sponge. Success is defined by a clean bowl and no drops or excessive force.

**Place Fruits:** A juice cup is positioned to the right of a plate and a banana to the left. The right arm must first grasp and place the cup onto the plate, and then the left arm must grasp and place the banana beside it. Success is achieved when both the cup and banana rest stably in their intended positions on the plate.

**Prepare Breakfast:** A cola bottle, an empty cup, an orange, and a banana are arranged. In Phase 1 the robot pours water from the bottle into the cup; in Phase 2 it places the orange and banana onto a plate; in Phase 3 it lifts and carries the loaded plate to a target area. Success requires each phase to complete without spillage, misplacement, or dropping.

## C Pseudo-code for SafeBimanual

**Algorithm Workflow.** At runtime (Algorithm 1), SafeBimanual first decomposes the task into discrete stages  $\{s_t\}$ , each governed by a termination condition. For simple tasks, we pre-select all stage-specific safety constraints and their associated termination checks, automatically switching to the next stage’s cost guidance when its condition is met. For long-horizon tasks, we instead re-invoke the VLM at each stage boundary to generate fresh safety primitives—adding minimal delay but enhancing adaptability to dynamic scenes.

Once the active stage  $s_t$  is determined, SafeBimanual applies CoT-VLM reasoning to the current observation  $\mathcal{O}_t$  and keypoints  $\mathcal{P}$  to identify the prevailing unsafe interaction pattern. The Safety Cost Scheduler uses this result to produce a binary mask  $\alpha$  that activates only the relevant cost terms  $\{\mathcal{C}_i\}$ . We then sample initial noise  $A_t^K$  and run DDIM denoising from  $k = K$  down to 1. To balance exploration and safety, we inject cost-gradient guidance only in the final  $M$  steps ( $k \leq M$ ); early steps remain pure denoising to preserve trajectory diversity, while later guidance steers the trajectory toward the safe output  $A_t^{\text{safe}}$  once noise levels are sufficiently low.

## D Implementation Details

### D.1 Real World Setup

All experiments use the Galaxea R1 humanoid robot, featuring two 7-DoF arms (70 cm reach, 100 N force), a 4-DoF waist, as shown in Figure 2. Scene rgb perception is provided by a ZED head-camera and wrist-mounted Intel D435i camera, supplemented by an Intel L515 for high-fidelity point clouds. The real-world experiments are performed on a single RTX 4080s GPU.

---

**Algorithm 1** Safety-Guided Sampling Process for Bimanual Manipulation

---

**Require:** Pretrained diffusion-based policy  $\epsilon_\theta$  [1], safety costs  $\{\mathcal{C}_i\}$ , taxonomy  $\mathcal{S}$ , keypoints  $\mathcal{P}$ , scheduler VLM [2, 3], total denoising steps  $K$ , guided denoising steps  $M$ , execution horizon  $m$

```
1:  $t \leftarrow 1$ 
2: while  $t \leq T$  do
3:    $s_t \leftarrow$  current task stage
4:    $\hat{p}_t \leftarrow \arg \max_{p \in \mathcal{S}} \Pr(p \mid \mathcal{O}_t, \mathcal{P}, s_t)$  ▷ CoT-VLM unsafe pattern
5:    $\alpha \leftarrow \text{Scheduler.VLM}(\hat{p}_t, \mathcal{P}, s_t)$  ▷ Cost-term mask
6:   sample initial noise  $A_t^K \sim \mathcal{N}(0, I)$ 
7:   for  $k = K$  down to 1 do ▷ DDIM denoising
8:      $\mu \leftarrow \mu(A_t^k, \mathcal{O}_t, k)$ 
9:      $A_{0|k} \leftarrow$  Estimate clean chunk via  $\epsilon_\theta$ 
10:    if  $k \leq M$  then ▷ apply safety guidance in final  $M$  steps
11:       $\mathcal{C}_{\text{sched}} \leftarrow \sum_i \alpha_i \mathcal{C}_i(A_{0|k}, \mathcal{P}, s_t)$ 
12:       $A_t^{k-1} \leftarrow \mu - \rho_k \nabla_{A_k} \mathcal{C}_{\text{sched}} + \sigma_k z, z \sim \mathcal{N}(0, I)$ 
13:    else
14:       $A_t^{k-1} \leftarrow \mu + \sigma_k z, z \sim \mathcal{N}(0, I)$ 
15:    end if
16:  end for
17:  Execute  $A_t^0$  for the next  $m$  steps
18:   $t \leftarrow t + m$ 
19: end while
```

---

## D.2 Training Settings of Baseline Methods

The key training setup for our baseline policies is detailed in Table 1. In simulation experiments, we evaluate 2D Diffusion Policy (DP), 3D Diffusion Policy (DP3), and RDT-1b; for real-world tasks, we use DP and improved 3D Diffusion Policy (iDP3) as baselines. All models are trained or fine-tuned with 50 episodes per task.

Table 1: Hyper-parameter Settings for DP, DP3, RDT-1b, and iDP3 Algorithms.

Parameter	DP [4]	DP3 [5]	RDT-1b [6]	iDP3 [7]
horizon	8	8	64	16
n.obs_steps	3	3	2	2
n.action_steps	6	6	30	15
num.inference_steps	100	10	5	10
dataloader.batch_size	128	256	32	64
dataloader.num_workers	0	8	8	8
dataloader.shuffle	True	True	True	True
dataloader.pin_memory	True	True	True	True
dataloader.persistent_workers	False	False	False	False
optimizer.target	AdamW	AdamW	AdamW	AdamW
optimizer.lr	1.0e-4	1.0e-4	1.0e-4	1.0e-4
optimizer.betas	[0.95, 0.999]	[0.95, 0.999]	[0.9, 0.999]	[0.95, 0.999]
training.lr.scheduler	cosine	cosine	constant	cosine
training.lr.warmup_steps	500	500	500	500
training.num_epochs	300	3000	10000	3000
training.gradient.accumulate.every	1	1	1	1
training.use.ema	True	True	True	True
training.gpu	6000 Ada	6000 Ada	A800	6000 Ada

## D.3 Implementation Details of Keypoint Proposal and Tracking

**Keypoint Proposal.** We refer to these keypoints as *safety constraint primitives*. In SafeBimanual, we define safety constraint primitives at both the object and gripper levels to capture all relevant interaction cues (see Figure 2):

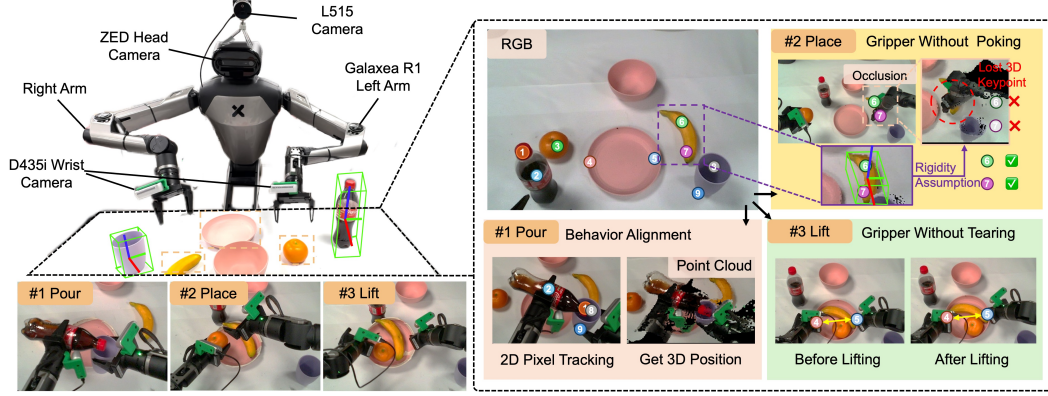


Figure 2: Real-world experimental platforms (Left) and Keypoint Proposal and Tracking (Right) in SafeBimanual.

- **Object-level keypoint:** Local surface keypoints are extracted from RGB-D images using DINOv2 features, SAM2 segmentation [8], PCA reduction,  $k$ -means clustering ( $k = 5$ ), and back-projection into world coordinates (see ReKeP [9] for more details). Candidates within 4cm of any existing keypoint are then filtered out to ensure spatial diversity. The object center and primary axis come from the translation and principal axis of the 6D pose estimated by Omni6DPose [10], with axis grounding as in OmniManip [11].
- **Gripper-level keypoint:** Gripper tip positions  $p_i^{\text{tip}} = \mathcal{F}(q^i) T_i^{\text{tip}}$  are computed via differentiable forward kinematics, where  $T_i^{\text{tip}}$  is the fixed offset from end-effector to tip. We do not perform surface segmentation or generate additional local keypoints on the gripper itself, which simplifies computation and analysis while still ensuring effective prevention of gripper collisions.

These object-level and gripper-level primitives are used to compute our five differentiable safety costs, enabling modular, interpretable, and dynamically scheduled safety constraints during diffusion-guided trajectory optimization.

**Keypoint Tracking Under Occlusion.** Although CoTracker3 [12] provides robust 2D pixel-level tracking under heavy occlusions, direct 3D point-cloud measurements can still be lost. To recover occluded 3D keypoints, we assume each local surface keypoint remains fixed in the object’s frame. Concretely, let  $T_{wo}(t) \in \text{SE}(3)$  be the object’s pose in the world frame at time  $t$  (from Omni6DPose [10]), and let  $p_k^o \in \mathbb{R}^3$  be the keypoint coordinates in that object frame. Then its world coordinate is  $k_t = T_{wo}(t) p_k^o$ . When first observed at  $t_0$ , we initialize  $p_k^o = T_{wo}(t_0)^{-1} k_{t_0}$ . For any later timestep  $t$  where the point is missing, we reconstruct it as  $\hat{k}_t = T_{wo}(t) p_k^o$ . If a noisy observation  $k_t^{\text{obs}}$  becomes available, we optionally refine via exponential smoothing:

$$p_k^o \leftarrow \beta p_k^o + (1 - \beta) T_{wo}(t)^{-1} k_t^{\text{obs}}, \quad \beta \in [0, 1].$$

This static-offset assumption ensures accurate, temporally consistent 3D keypoint estimates even under severe occlusions.

#### D.4 CoT-based Vision-Language Model Prompts

After identifying keypoint candidates, we number them  $\{0, \dots, K-1\}$  and overlay them on the key-frame RGB image, which is then input, along with a high-level task instruction, into the Vision-Language Model (VLM). We adopt a code-style prompt template centered around a `stage_n_llm_output` dictionary and a `stage_n_end()` function skeleton. Only minimal examples are given to illustrate stage structure and constraint formats; all remaining fields (e.g., number of stages, keypoints per stage, termination logic) are inferred autonomously by the VLM through physical priors and visual understanding.

In this work, GPT-4o is used as the default VLM. As newer VLMs emerge, they can be directly substituted to enhance performance without modifying downstream code. Hence, our goal is not

prompt engineering per task type, but to establish a sustainable, end-to-end pipeline: as long as key names and function interfaces remain consistent, newer VLMs can iteratively generate stage-wise constraints that conform to the template format and drive robotic execution in increasingly complex tasks. To illustrate the CoT-VLM scheduling process, we present a concrete prompt instance that aligns with our two-stage procedure:

```
## Instructions
## Universal VLM-Prompt Template
# =====
# Suppose you are helping a bimanual robot to ensure the safety
# during performing manipulation tasks by output a python dictionary
# and stage_end function. The manipulation task is given as an image
# of the environment, overlaid with keypoints marked with their
# indices, along with coordinates of keypoints. Your job should be
# focus on avoiding unsafe behaviors between the objects.

# Inputs: - system prompt(you are safety assistant)
#         - user RGB image(keypoints numbered)
#         - user np.array of keypoint XYZ coordinates

### ---- 1. KEYPOINT ALIAS TABLE -----
KEYPOINTS = {
    -1 : "left_gripper_tip",
    -2 : "right_gripper_tip",
    -3 : "apple_centre",
    -4 : ...
    # Additional ids are defined by the annotated image
}

# =====

## GENERAL NOTES
# - Keypoints coordinates input, is an np.array of three-dimensional
#   point coordinates, each row represents the X Y Z coordinates of a
#   point, the first row represents the point 1, the second represents
#   the point 2, and so on.
# - Choose points by rows, which is point 1, 2, 3... And -1 is
#   left_gripper_tip while -2 is right_gripper_tip
# - Decide which arm is nearer if only one should act.
# - Typical guidance types: poking / tear / align / gripper collision /
#   object collision.
# - stage_end is omitted for final stage unless needed.

## STEP-BY-STEP INSTRUCTION FOR GPT4o
# 1. Determine how many stages are involved in the task. Grasping
#    must be an independent stage.
Some examples:
    - "dual_bottle_easy":
      - 2 stages: "grasp bottles", "move and place bottles"
# 2. For each stage:
#    a. Choose guidance type & fill llm_output. (poking, objects-collision, grippers-
#       collision, tear, align)
#       if gripper is going to grasp object:
#           enable poking_guidance
#       if two objects are both grasped and moving:
#           enable collision_guidance (between objects)
#       if object is not successfully grasped:
#           enable collision_guidance (between grippers)
#       if both grippers are holding the same object:
#           enable tear_guidance
#       if two points on the different objects need to be aligned:
#           enable align_guidance

# llm_output example:
```

```

llm_output = {
    "enable_?_guidance":
    {
        "enable": True/False,
        "enable_left_arm":
        {
            "enable": True/False,
            "point": "?",
        },
        "enable_right_arm":
        {
            "enable": True/False,
            "point": "?",
        }
    },
}

# b. If the stage needs a finish condition, implement
def stage_N_end(self):
    ...
    return ...

# **Nnote:**
# - Consider use bool to determin the start and end of the stage, in
#   case the stage is enable more than once.
# c. Write one-line comment explaining key-point choices.
# 3. Preserve ordering: Stage 1,2,3...
# 4. Return only code no extra narration.

# =====

## Example A "apple handover"
# Stages: 1 grasp apple, 2 handover
### stage_1 "grasp apple"
# left arm nearer to apple (choose apple centre)
llm_output = {
    "enable_poking_guidance":{
        "enable": True,
        "enable_left_arm": {"enable": False, "point": "-3"},
        "enable_right_arm": {"enable": True, "point": null}
    }
}

def stage_1_end(self):
    return self.is_left_gripper_close() and (self.apple_pose[2] > 0.85)

# both grippers hold apple centre; tips are "-1" (L) & "-2" (R)
llm_output_2T = {
    "enable_tear_guidance":{
        "enable": True,
        "enable_left_arm": {"enable": True, "point": "-1"},
        "enable_right_arm": {"enable": True, "point": "-2"}
    }
}

# both grippers hold apple centre; tips are "-1" (L) & "-2" (R)
llm_output_2F = {
    "enable_gripper_collision_guidance":{
        "enable": True,
        "enable_left_arm": {"enable": True, "point": "-1"},
        "enable_right_arm": {"enable": True, "point": "-2"}
    }
}

def stage_tear_guidance_start(self):
    return self.is_left_gripper_close() and self.is_right_gripper_close() and (self.
apple_pose[2] > 0.85)

```

```

def stage_gripper_collision_guidance_start(self):
    return (self.is_left_gripper_close() and self.left_gripper.pose()[2] > 0.85 and
            self.apple_pose[2] < 0.8)

def get_current_guidance(self, llm_output_2T, llm_output_2F):
    # Decide which guidance dictionary to return
    if self.stage_tear_guidance_start():
        return llm_output_2T
    elif self.stage_gripper_collision_guidance_start():
        return llm_output_2F

## Example B "pour coke into cup"
# Stages: 1 grasp objects, 2 align, 3 pour
### stage_1 "grasp coke and cup"
# left gripper: kp-3 (coke mid), right: kp-5 (cup mid)
llm_output = {
    "enable_poking_guidance":{
        "enable": True,
        "enable_left_arm": {"enable": True, "point": "3"},
        "enable_right_arm": {"enable": True, "point": "5"}
    }
}

def stage_1_end(self):
    return self.left_gripper_close() and self.right_gripper_close()

### stage_2 "align coke with cup"
# collision points kp-1 (coke neck) & kp-4 (cup rim)
llm_output = {
    "enable_collision_guidance":{
        "enable": True,
        "enable_left_arm": {"enable": True, "point": "1"},
        "enable_right_arm": {"enable": True, "point": "4"}
    }
}

def stage_2_end(self):
    height = abs(self.object1_pose[2] - self.object2_pose[2])
    return height > 0.10 # coke 10cm above cup

### stage_3 "pour coke"
# align guidance kp-1 & kp-4 (same as above)
llm_output = {
    "enable_align_guidance":{
        "enable": True,
        "enable_left_arm": {"enable": True, "point": "1"},
        "enable_right_arm": {"enable": True, "point": "4"}
    }
}

# ----- END OF EMPTY TEMPLATE -----

```

As an example, Figure 2 illustrates a sub-task of pouring water. After prompting, the Vision-Language Model (VLM) analyzes a keyframe image with keypoints mask and correctly infers the high-risk scenario of **Objects Behavior Misalignment**, activating the cost term  $\mathcal{C}_2$ . Based on its interpretation, the VLM identifies and returns two keypoints: one on the top of the bottle and one on the top of the cup. To initiate a safe and effective pouring motion, a vector  $\vec{a}$  is first constructed between the top point of the cup and the central axis point of the cup. This vector is designed to have the minimum angle with the world frame's vertical z-axis. Next, a vector  $\vec{b}$  is constructed between the top of the cup and the top of the bottle. The robot then executes the pouring motion by aligning  $\vec{b}$  with  $\vec{a}$ , thereby ensuring the bottle is tilted in a direction that minimizes spillage and avoids collision.



This behavior alignment cost ensures that dynamic constraints are grounded in visual observations and can be flexibly scheduled based on scene semantics and geometry.

## E Additional Results

### E.1 Visualization of the Generated Safe Bimanual Trajectories in RoboTwin

In Figure 3, we compare Diffusion Policy outputs against trajectories refined by SafeBimanual across three RoboTwin tasks. **Block Handover.** During the handover stage, SafeBimanual correctly identifies the risk of gripper tearing and activates  $\mathcal{C}_4$  to maintain a safe inter-tip distance. As a result, the optimized trajectory avoids the tearing seen in the unmodified Diffusion Policy rollout. **Blocks Stack (Easy).** At the pick stage, SafeBimanual detects potential gripper poking and applies  $\mathcal{C}_3$  to enforce directional alignment between the gripper tip and the cube keypoint. This prevents surface damage and enables a stable grasp. **Dual Shoes Place.** While lifting the shoes, SafeBimanual recognizes an imminent object-object collision and selects the shoe-head (left) and shoe-heel (right) keypoint pair to compute  $\mathcal{C}_1$ . The guided trajectory steers clear of high-risk regions, ensuring both shoes are moved safely to their target areas.

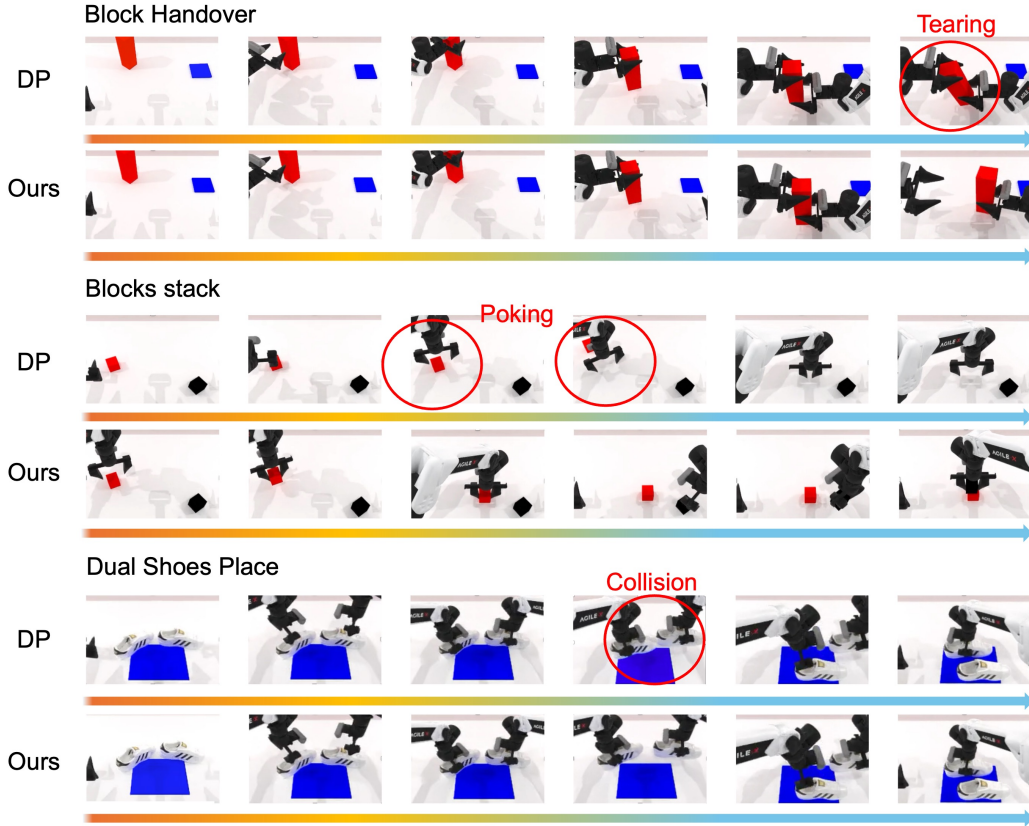


Figure 3: Simulation Tasks Visualization.

### E.2 Visualization of the Generated Safe Bimanual Trajectories in Real-world

Figure 4, we compare Diffusion Policy outputs against trajectories refined by SafeBimanual across three real-world tasks: **Pour Water.** During the pouring phase, SafeBimanual infers misalignment and activates  $\mathcal{C}_2$ , selecting the bottle mouth and cup mouth keypoints. The guided trajectory corrects the tilt to align spout and cup, preventing spillage. **Pass Banana.** In the handover stage, SafeBimanual detects potential tearing and enables  $\mathcal{C}_4$ , enforcing a fixed inter-tip distance as both grippers hold the banana. This avoids relative motion that would damage the fruit. **Place Fruits.** At the



final placement phase, SafeBimanual triggers the object–object collision cost  $C_1$ , choosing a surface keypoint on the banana and one on the cup. The resulting trajectory maintains safe separation, preventing the banana from knocking over the juice cup.

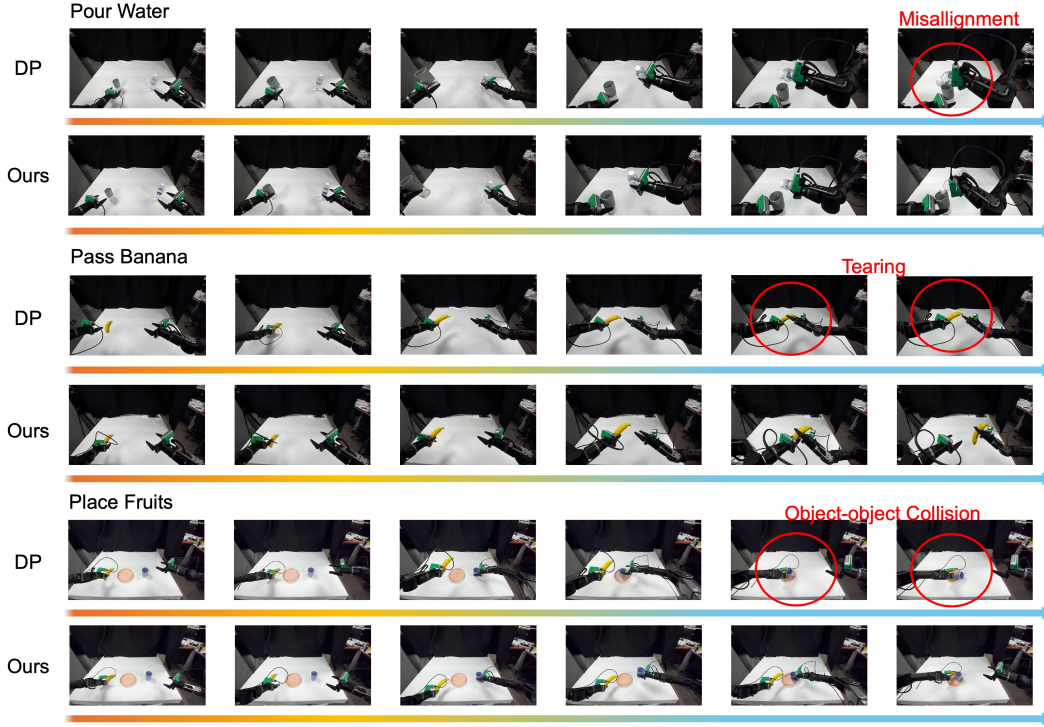


Figure 4: Real-world Tasks Visualization.

## References

- [1] J. Yu, Y. Wang, C. Zhao, B. Ghanem, and J. Zhang. Freedom: Training-free energy-guided conditional diffusion model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 23174–23184, 2023.
- [2] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [3] R. Zhang, B. Zhang, Y. Li, H. Zhang, Z. Sun, Z. Gan, Y. Yang, R. Pang, and Y. Yang. Improve vision language model chain-of-thought reasoning. *arXiv preprint arXiv:2410.16198*, 2024.
- [4] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.
- [5] Y. Ze, G. Zhang, K. Zhang, C. Hu, M. Wang, and H. Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. *arXiv preprint arXiv:2403.03954*, 2024.
- [6] S. Liu, L. Wu, B. Li, H. Tan, H. Chen, Z. Wang, K. Xu, H. Su, and J. Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.
- [7] Y. Ze, Z. Chen, W. Wang, T. Chen, X. He, Y. Yuan, X. B. Peng, and J. Wu. Generalizable humanoid manipulation with 3d diffusion policies. *arXiv preprint arXiv:2410.10803*, 2024.

- [8] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, et al. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024.
- [9] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*, 2024.
- [10] J. Zhang, W. Huang, B. Peng, M. Wu, F. Hu, Z. Chen, B. Zhao, and H. Dong. Omni6dpose: A benchmark and model for universal 6d object pose estimation and tracking. In *European Conference on Computer Vision*, pages 199–216. Springer, 2024.
- [11] M. Pan, J. Zhang, T. Wu, Y. Zhao, W. Gao, and H. Dong. Omnimanip: Towards general robotic manipulation via object-centric interaction primitives as spatial constraints. *arXiv preprint arXiv:2501.03841*, 2025.
- [12] N. Karaev, I. Makarov, J. Wang, N. Neverova, A. Vedaldi, and C. Rupprecht. Cotracker3: Simpler and better point tracking by pseudo-labelling real videos. In *Proc. arXiv:2410.11831*, 2024.