Supplementary Material

Table of Contents

A Implementation Details

A.1 Task Construction

Domain Randomization. To enhance the robustness and generalization capability of our system, we implement comprehensive domain randomization strategies during the environment reset phase. The randomization encompasses multiple aspects of the environment:

- Object Mass Randomization: At the beginning of each episode, object masses are randomized by scaling each object's default mass with a random factor sampled from a uniform distribution $U(1, 1.5)$ (units: kg):

$$m_{\text{curr}} = m_{\text{default}} \cdot \alpha, \quad \alpha \sim U(1, 1.5)$$

- Object Position Randomization: Small perturbations are applied to the initial positions of objects to introduce variability (units: meters):

$$\Delta x \sim U(-0.02, 0.02)$$
$$\Delta y \sim U(-0.02, 0.02)$$

- Target Position Randomization: The initial position of the target object within the box is randomized. The random displacements are sampled from (units: meters):

$$\Delta x \sim U(-0.15, 0.15)$$
$$\Delta y \sim U(-0.2, 0.2)$$

  This ensures that the target object can be placed within 70% of the box's area.

- Camera Mount Randomization: During data collection, the camera's mounting position is perturbed with small random displacements (units: meters):

$$\mathbf{p}_{\text{camera}} = \mathbf{p}_{\text{default}} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim U(-0.01, 0.01)^3$$

A.2 Retrieval Policy Training

Reward Design. We carefully design the reward for our system, which contain serval item including distance reward, stir reward, proximity clearance reward, pixel emergency reward and penalty. We show the specific parameters setup for each reward item in Table 4.

Table 4: Reward Components and Definitions.

Reward Item	Definition	Parameters
Distance Reward	$r_{\text{dist}} = \exp(-5 \cdot \min(d - e_0, 0))$	$e_0 = 0.15$
Stir Reward	$r_{\text{stir}} = \alpha\|p^{all} - p^{all}_{t-1}\|_2$	$\alpha = 500.0$
Proximity Clearance Reward	$r_{\text{clean}} = \beta \cdot r_{\text{dist}} \cdot \sum_{i=1}^{k} f_i$	$\beta = 100.0$
Pixel Emergency Reward	$r_{\text{pixel}} = C/15$	-
Penalty	$\lambda_1 \|\|a_t^{\text{hand}} - a_{t+1}^{\text{hand}}\|\|_2^2 + \lambda_2 \|\|a_t^{\text{arm}}\|\|_2^2$	$\lambda_1 = 1.0$, $\lambda_2 = 1.0$

Policy Training. We employ the Proximal Policy Optimization (PPO) algorithm [28] to train a continuous control policy using an actor-critic architecture. Detailed hyperparameters are provided in Table 5. The policy network is parameterized as a multi-layer perceptron (MLP) with three layers of sizes [256, 256, 256], utilizing the ELU activation function for improved gradient flow and non-linearity. The standard deviation of the policy distribution is learned via a log-std representation, enabling dynamic adjustment of exploration during training.

Table 5: Hyperparameters for PPO Training

Category	Parameter	Value	Description
Model Architecture	MLP Layers	[256, 256, 256]	Number of neurons per layer
	Activation Function	ELU	Non-linearity used in the network
Training Parameters	Learning Rate	3×10^{-4}	Step size for policy update
	Discount Factor (γ)	0.99	Reward discounting factor
	GAE Parameter (τ)	0.95	Smoothing factor for GAE
	Entropy Coefficient	0	Weight of entropy regularization
	Gradient Clipping	Norm 1, Truncation Enabled	Prevents gradient explosion
	Clip Range (ϵ)	0.1	PPO clipping threshold
	KL Threshold	0.02	KL divergence threshold for stopping training
	Minibatch Size	512	Batch size for optimization
	Mini Epochs	5	Number of updates per batch
	Horizon Length	8	Number of steps before update
	Max Training Epochs	50,000	Maximum number of training iterations
	Value Learning Rate	1×10^{-3}	Learning rate for value function

To ensure stable and efficient learning, we adopt adaptive learning rate scheduling, starting at 3×10^{-4}. The advantage function is normalized to reduce variance in policy gradient updates, while the Generalized Advantage Estimation (GAE) [27] parameter τ is set to 0.95, striking a balance between bias and variance. Gradient clipping with a norm threshold of 1 is applied to prevent exploding gradients. To constrain policy updates, we employ a PPO clipping range of 0.1, which limits large deviations from the current policy, and enforce a KL divergence threshold of 0.02 to promote conservative updates and prevent policy collapse.

The training runs for a maximum of 50,000 epochs, with model checkpoints saved every 1,000 epochs. The best-performing model is selected based on validation returns and retained after 200 epochs to prevent overfitting. A separate centralized value function is used for advantage estimation, parameterized as an MLP with the same architecture as the policy network. The critic network employs a higher learning rate of 1×10^{-3} to facilitate faster convergence in value estimation, a choice informed by preliminary experiments indicating more stable critic updates with this configuration.

A.3 Spatial-aware Representation Training

Model Architecture. We design a hierarchical GNN model for estimating visibility and extracting structured representations of target objects in cluttered scenes. The model integrates a PointNet++-based encoder for each object category (target, hand, obstacles) and performs message passing via EdgeConv [31] layers. Specifically, three main components constitute the model: (1) **Point Feature Encoding:** Each point cloud (target, hand, obstacles) is encoded via a PointNet++ module [26], consisting of three Set Abstraction (SA) layers and two Feature Propagation (FP) layers. Global max pooling over the final layer provides a fixed-length feature representation. (2) **Graph Construction:** For each sample, a complete graph is built among all encoded nodes, including one target node, one hand node, and $K = 9$ nearest neighbor obstacles. (3) **Graph Message Passing:** Two layers of EdgeConv are applied to propagate features among nodes, followed by a third EdgeConv that aggregates pairwise edge features for visibility prediction.

Implementation Details. Before training this module, we collect 2000 trajectories generated by a basic reinforcement learning policy. The policy is trained using reward signals. We optimize the model using mean squared error (MSE) loss, training the network to predict the visibility label, defined as the number of pixels occupied by the target object in the top-view camera image. The dataset is randomly split into 90% for training and 10% for validation. We employ the Adam optimizer and apply a learning rate scheduler that reduces the learning rate upon plateauing of the validation loss. To monitor early-stage performance closely, validation is performed every 29 training iterations.

To support efficient parallel data loading and training, we design a function to batch dynamic obstacle sets and filter invalid data samples during runtime. Each episode in the dataset contains 209 time steps, and the model is trained either on the last step or all time steps, depending on the configuration. Table 6 summarizes all hyperparameters used for GNN training. The output dimensionality of the extracted structured feature (used in downstream RL) is set to $d = 32$ unless otherwise specified.

Table 6: Hyperparameters for GNN Training.

Category	Parameter	Value	Description
Model Architecture	PointNet++ Encoder Output Dim	64	Dimension of point cloud encoding
	GNN Hidden Dim	64	Hidden units in EdgeConv layers
	Final Output Dim	32	Output dim of GNN feature extractor
	Num of Neighbors (K)	9	Number of obstacle nodes in graph
PointNet++ Settings	SA Layer Config	[512, 128, 1]	Points per SA layer
	SA Radii	[0.2, 0.4, all]	Search radii for each SA layer
	MLPs per SA Layer	[64,64,128], [128,128,256], [256,512,out]	MLPs applied in SA layers
	Graph Convs	2 EdgeConv + 1 predictor	Number and type of GNN layers
Training Parameters	Optimizer	Adam	Optimization algorithm
	Learning Rate	1×10^{-4}	Initial step size
	LR Scheduler	ReduceLROnPlateau	Decay LR on validation plateau
	Scheduler Settings	factor 0.5, patience 5	Scheduler configuration
	Loss Function	0.5×MSE + 0.5×Huber	Objective function
	Batch Size	32	Samples per batch
	Epochs	30	Max training epochs
Validation Settings	Validation Interval	Every 29 steps	Frequency of validation
	Data Split	90% train / 10% val	Dataset partition ratio
	Time Sequence Mode	`last` (default)	Sampling strategy from sequence

A.4 Sim-to-Real transfer

For real-world deployment, we collect a set of trajectories generated by our RL expert policy. To ensure data quality and consistency, we first select successful trajectories. We then filter out trajectories where the finger's z-coordinate lower 2 cm above the box, a threshold empirically chosen to prevent unstable behavior and reduce the risk of collision with the box during manipulation. To promote generalization, we balance the dataset across various target object positions within the box, ensuring uniform coverage of spatial configurations. This prevents the model from overfitting to specific object placements and enhances its adaptability to unseen scenarios.

Figure 5 shows the model architecture, which consists of a state encoder followed by a multi-head self-attention mechanism with six transformer layers, each containing six attention heads. This design captures complex temporal dependencies across historical state sequences of length 15, enabling accurate prediction of future actions over a 5 step horizon chunking. The hidden dimension of 384, paired with a feed-forward expansion ratio of 5.33 (2048/384), strikes a balance between model expressiveness and computational efficiency.

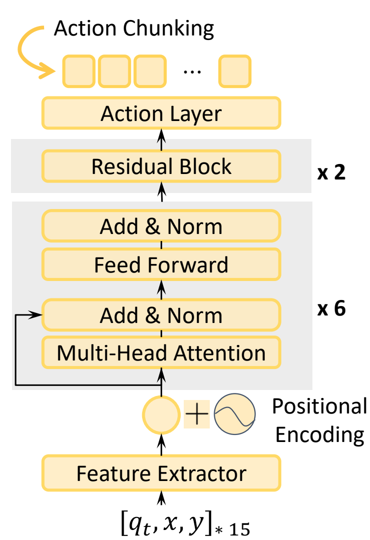

Figure 5: The architecture of student policy.

To effectively manage the continuous action space inherent in robotic control, we introduce a custom Negative Log Product Loss function, which penalizes trajectory deviations more sensitively than traditional mean squared error. This loss function emphasizes multi-step consistency, enhancing the model's predictive stability. Training is performed using the Adam optimizer with a learning rate of 1×10^{-4} over 10,000 iterations and a batch size of 512. Mixed-precision training accelerates computation without compromising accuracy, while gradient clipping at 1.0 maintains stable learning dynamics. Hyperparameter selection was guided by cross-validation on a held-out dataset to optimize both performance and robustness. Detailed architectural specifications and hyperparameters are provided in Table 7. Despite strong simulation performance, real-world deployment introduces challenges such as sensor noise, domain discrepancies, and dynamic environmental conditions. Our transformer model mitigates these issues by leveraging temporal patterns to predict smooth and consistent actions.

Table 7: Hyperparameters of Distilled Policy.

Category	Parameter	Value	Description
Model Architecture	Input State Dimension	9	Size of input state vector
	Action Dimension	13	Number of output actions
	History Frames	15	Past frames used as input
	Future Action Frames	5	Future actions predicted
	Transformer Hidden Size (d_{model})	384	Hidden layer size
	Number of Attention Heads	6	Transformer attention heads
	Number of Transformer Layers	6	Transformer depth
	Feed-forward Dimension	2048	FFN hidden size
	Dropout Rate	0.15	Dropout probability
Training Parameters	Batch Size	512	Training batch size
	Total Iterations	10,000	Training iterations
	Learning Rate	1e-4	Initial learning rate
	Optimizer	Adam	Optimization algorithm
	Loss Function	Negative Log Product Loss	Loss function used
	Gradient Clip Norm	1.0	Gradient clipping threshold

A.5 Baseline Implementation

In our simulation experiments, we compare our method against several baseline approaches. *Ours (SAC)* is a variant of our system where the PPO algorithm is replaced with Soft Actor-Critic (SAC) to evaluate the impact of different RL optimization strategies. *Ours (Gripper)* is a variant of our system where the dexterous hand is replaced with a UMI gripper to assess the importance of hand morphology in cluttered object retrieval. The other two baselines are *Visual-based Motion Planning Search (VMP)* and *Grasp-Pick*. VMP is a heuristic motion planning approach that uses target object segmentation masks to guide the robotic hand toward the target object and employs predefined rules for retrieval manipulation. Grasp-Pick involves sequentially grasping and placing objects based on the support relationships within the cluttered scene.

A.5.1 Ours (SAC)

SAC is an off-policy reinforcement learning algorithm grounded in the maximum entropy framework, which promotes exploration by encouraging stochastic action selection. The algorithm comprises a soft Q-function $Q_\theta(s, a)$ and a stochastic policy $\pi_\phi(a|s)$.

The soft Q-function is defined as the expected cumulative return:

$$Q_\theta(s_t, a_t) = \mathbb{E}\left[\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \,\middle|\, s_t, a_t\right], \tag{5}$$

where $\gamma \in [0, 1]$ denotes the discount factor. In maximum entropy RL, this return is typically augmented with an entropy bonus, though omitted here for clarity.

The Q-function parameters θ are optimized by minimizing the soft Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t, s_{t+1}, r_t)\sim\mathcal{B}}\left[\left(Q_\theta(s_t, a_t) - r_t - \gamma \bar{V}(s_{t+1})\right)^2\right], \tag{6}$$

where $\bar{V}(s_{t+1}) = \mathbb{E}_{a\sim\pi_\phi}\left[Q_{\bar{\theta}}(s_{t+1}, a) - \alpha \log \pi_\phi(a|s_{t+1})\right]$ uses a target Q-network with parameters $\bar{\theta}$. The temperature α controls the strength of the entropy regularization and can be fixed or learned during training.

The policy parameters ϕ are updated by minimizing:

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{B}, a_t\sim\pi_\phi}\left[\alpha \log \pi_\phi(a_t|s_t) - Q_\theta(s_t, a_t)\right]. \tag{7}$$

SAC alternates between policy evaluation and improvement, enabling stable and sample-efficient learning. In this work, we adopt SAC as our base reinforcement learning algorithm due to its robustness in continuous control tasks.

Table 8: Hyperparameters for SAC Training

Category	Parameter	Value	Description
Network Architecture	Actor Hidden Dim	256	Hidden layer dimension of actor network
	Actor Hidden Depth	3	Number of hidden layers in actor network
	Critic Hidden Dim	256	Hidden layer dimension of critic network
	Critic Hidden Depth	3	Number of hidden layers in critic network
Training	Actor Learning Rate	5e-4	Learning rate for actor network
	Critic Learning Rate	5e-4	Learning rate for critic network
	Temperature Learning Rate	0.05	Learning rate for entropy coefficient
	Batch Size	8192	Number of samples per training batch
	Max Gradient Norm	0.5	Maximum gradient norm for clipping
	SAC Epochs	8	Number of training epochs per update
	Discount Factor	0.99	Future reward discount factor
	Initial Temperature	0.1	Initial entropy coefficient
	Critic Tau	0.05	Soft update coefficient for target critic
	Learnable Temperature	True	Whether to learn entropy coefficient
	Number of Environments	512	Parallel environments in IsaacGym
	Update Interval	1	Environment steps between updates
	Seed Steps	32	Initial random steps per environment
	TD n-step	3	Number of steps for TD learning
	Replay Buffer Size	5M	Maximum capacity of experience replay

We implement the baseline *Ours (SAC)*, incorporating several key modifications to improve training efficiency and stability. The implementation is built on the IsaacGym physics engine, enabling large-scale parallelism with 512 simulated environments to accelerate data collection. The full set of hyperparameters is listed in Table 8. Both the actor and critic are modeled as three-layer multilayer perceptrons, each with 256 hidden units per layer. The actor outputs the mean and log standard deviation of a Gaussian action distribution, with the log standard deviation constrained between -5 and 2 to ensure numerical stability. The critic adopts a double Q-learning architecture with target networks updated via a soft update rule using $\tau = 0.05$.

To stabilize training, we use a large batch size of 8192 and apply policy updates at every timestep. At the start of training, each environment performs 32 random actions to populate the replay buffer. The actor and critic are optimized using the AdamW optimizer with a learning rate of 5e-4, while the temperature parameter is updated with a learning rate of 0.05. Gradient clipping with a threshold of 0.5 is applied to mitigate exploding gradients. The replay buffer stores up to 5 million transitions. For each policy update, we perform 8 epochs of optimization using a single mini-batch per epoch, promoting efficient data usage while reducing the risk of overfitting. The temperature parameter is initialized to 0.1 and is learned adaptively to maintain a balance between exploration and exploitation.

Our parallelized training framework significantly enhances sample throughput by synchronously simulating 512 environments, all of which share a single policy network. This design ensures consistent policy updates while leveraging high-throughput simulation. Additionally, we adopt 3-step temporal difference (TD) learning to reduce bias in value estimation and improve overall policy performance.

A.5.2 VMP

The VMP system implements a vision-guided manipulation framework for dexterous robotic retrieval tasks in cluttered environments. It integrates visual perception, motion planning, and control execution through a state machine architecture to ensure reliable object manipulation.

The vision module employs a top-down camera with a resolution of 1024×512, capturing RGB, depth, and segmentation maps of the workspace. Target objects are identified using segmentation masks obtained from the segmentation map, with their IDs corresponding to known object labels. The center of the target mask's bounding box is extracted as the 2D image coordinate, which is projected into 3D space using depth data to obtain precise object localization. For motion planning, the robotic arm moves its end effector to the computed 3D coordinate and performs a scrape action to retrieve the object.

When the target object is completely occluded and its segmentation mask cannot be detected, the system employs an exploration strategy by randomly sampling four 3D coordinates within the cluttered bin area. The arm sequentially moves to these coordinates, performing scrape actions to uncover the target object. Specifically, the entire motion planning and scrape action process employs a four-stage approach to ensure reliable object retrieval.

Pre-approach stage: The end-effector moves to a predefined position ($h = 0.5\,\text{m}$) above the target object. This configuration facilitates subsequent control of the hand to reach any position within the bounding box.

Final approach stage: Precise positioning is achieved using visual feedback combined with damped least squares inverse kinematics:

$$\tau = J^T(JJ^T + \lambda I)^{-1}\Delta x$$

where $\lambda = 0.05$ is the damping parameter, J is the Jacobian matrix, and Δx represents the positional error.

Scraping stage: The system executes a periodic motion pattern defined by:

$$x(t) = A\sin(2\pi ft + \phi) + O$$

where the amplitude $A = 2.0$, frequency $f = 20\,\text{Hz}$, phase shift $\phi = \pi/4$, and offset $O = 0.5$.

Reset stage: The target object has been retrieved, so the robot arm will return the end effector to its initial position.

The control execution module utilizes position-based control for both arm and finger joints. Adaptive damping parameters are applied to ensure stable motion, while joint limits are strictly enforced throughout the execution process: $q_{\text{min}} \leq q \leq q_{\text{max}}$.

A.5.3 Grasp-Pick

This method relies on support relationships among cluttered objects to guide the grasping sequence. To ensure these assumptions hold, we designed tailored setups for both simulation and real-world experiments.

In simulation, object positions are directly accessible, enabling precise calculation of support relationships. We employ a KD-Tree [29] to organize the coordinates of objects near the target. Based on Euclidean distance, we select the three to five nearest objects, depending on the scenario, and manipulate them sequentially to clear access to the target. While this approach offers computational simplicity, it assumes ideal sensing conditions and may not generalize to more complex spatial arrangements.

For robotic control, we implement damped least squares inverse kinematics:

$$\dot{\mathbf{q}} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \lambda\mathbf{I})^{-1}\dot{\mathbf{x}},$$

where λ is the damping coefficient, $\mathbf{J}$ is the Jacobian matrix, and $\dot{\mathbf{x}}$ is the desired end-effector velocity. This formulation offers stable solutions near singularities but may limit the dexterity needed in cluttered environments.

In real-world experiments, sequentially grasping and removing multiple objects in stacked scenes with a dexterous hand remains challenging due to perception and control limitations. To address this, we employ predefined trajectories for each trial, simulating an idealized execution scenario. While this implementation provides an upper-bound estimate of this method's efficiency, it does not reflect the challenges of autonomous execution in unstructured environments.

A.6 Evaluation metric

Exposure Calculation. The primary goal of object retrieval is to locate the target object and enhance its visibility within the camera's field of view, facilitating subsequent manipulations. We define *exposure* as the proportion of unobstructed pixels of the target object in the imaging plane. Considering that changes in the object's pose can affect the number of visible pixels, we proceed as follows:

At timestep t, we record the target object's visible pixels p_t^{curr} and its 6D pose. Subsequently, all objects except the target are removed, the target object's recorded 6D pose is reset, and its total visible pixels are recorded as p_t^{all}. The exposure at time t is then computed as:

$$\text{exposure}_t = \frac{p_t^{\text{curr}}}{p_t^{\text{all}}}. \tag{8}$$

Figure 6: **Examples of successful and failed object retrievals on the real robot.**

Success in Real-World Experiments. To systematically evaluate the success rate of object retrieval on the real robot, we capture images before and after each task using a side-mounted RealSense D435 camera. The success of the retrieval is determined by comparing the exposure of the target object in these images. As illustrated in Figure 6, we present examples of both successful and failed retrieval attempts.

A.7 Computing Resources

All experiments are conducted on a single NVIDIA RTX 4090 GPU. The reinforcement learning training takes approximately 6 hours, the GNN module requires around 20 minutes, and the student policy training completes within 4 minutes.

B Additional Experiments

B.1 Impact of Occlusion Rate

We also explore the impact of clutter occlusion rate (i.e., $1 -$ exposure) on target objects. Figure 7 presents the relationship between occlusion rates, retrieval success rate (RSR), and retrieval steps (RS) in cluttered environments, comparing smaller and larger target objects. As the occlusion rate rises from 30% to 100%, RSR decreases for both object sizes, with smaller objects consistently achieving higher RSR than larger ones. This indicates that larger objects, although

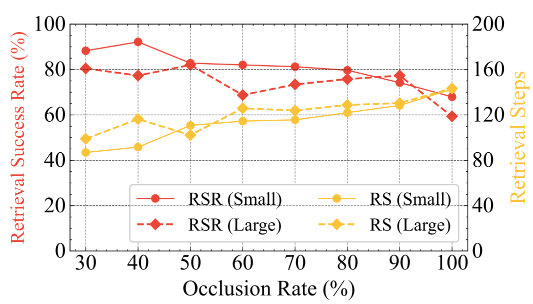

Figure 7: **Impact of Occlusion Rate on Performance and Efficiency.** We evaluate the retrieval success rate and retrieval steps of our policy for small and large target objects under varying occlusion levels.

generally easier to detect, are more prone to being significantly obstructed by substantial clutter. In contrast, retrieval steps increase as occlusion rates rise, signifying reduced efficiency in highly cluttered settings. Smaller objects generally require fewer retrieval steps than larger ones across most occlusion levels, likely because retrieving larger objects involves removing more obstructing clutter. These findings reveal that smaller objects are retrieved with higher success and efficiency, while larger objects face greater challenges posed by occlusion.