AutoIntervene: Calibrated Intervention for Action-Chunking Imitation Learning Policies

***Abstract*— Action-chunking visuomotor policies learn from demonstrations and improve temporal consistency by predicting short action sequences rather than single-step commands. Yet perception errors and execution drift can move the robot outside the demonstration distribution, while the policy continues to produce smooth action chunks that are inconsistent with the observed state. We present *AutoIntervene*, an online framework that selectively transfers control between an action-chunking policy and an operator during deployment. *AutoIntervene* evaluates proposed chunks against a visual-action support memory built from successful task executions, combining visual similarity with consistency between proposed and reference actions. Phase-local support governs policy-to-operator transfer within the current task phase, whereas global support governs the return to policy control after operator recovery. We calibrate separate switching thresholds for the two directions from empirical quantiles of evaluation-level scores on held-out expert demonstrations, avoiding direct manual tuning of score cutoffs. Intervention segments retained from successful rollouts target learner-induced states and provide corrective supervision for subsequent policy updates. Experiments on real-world bimanual manipulation tasks show higher post-adaptation task success and lower operator-control time than manual intervention.**

I. Introduction

Imitation learning provides a practical framework for learning robot policies from demonstrations [1], including stable motion generators that adapt to environmental changes [2], while offline human data has enabled complex multi-stage and long-horizon manipulation [3], [4]. Recent action-chunking policies improve temporal consistency by predicting short sequences of future actions rather than single-step commands, enabling visuomotor systems to execute contact-rich manipulation tasks without hand-designed controllers [5]. However, these policies remain brittle under deployment-time distribution shift [6]. DART broadens demonstration coverage through noise injection but provides no deployment-time mechanism for detecting unsupported policy behaviour or requesting corrective control [7]. Small perception errors, missed contacts, or accumulated execution error can move the robot into states that are poorly covered by the demonstration data. Once this occurs, an action-chunking policy can continue producing smooth action chunks while no longer making task progress, leading to failures such as misaligned grasps or incomplete subtask transitions [8], [9].

In this paper, we introduce *AutoIntervene*, an online intervention framework for improving the deployment reliability of action-chunking policies. AutoIntervene evaluates each proposed action chunk against successful visual-action support and transfers control when the proposal repeatedly becomes unsupported. It uses phase-local references to decide when control should pass from the policy to the operator and global references to decide when it should return to the policy, allowing autonomy to resume from any supported task phase after operator correction. At each adaptation round, separate switching thresholds are calibrated offline by recomputing proposals and support scores on held-out successful expert demonstrations under the corresponding retrieval scope. Joint visual-action testing is important because visual familiarity alone cannot disambiguate repeated task phases [8], [10]–[12].

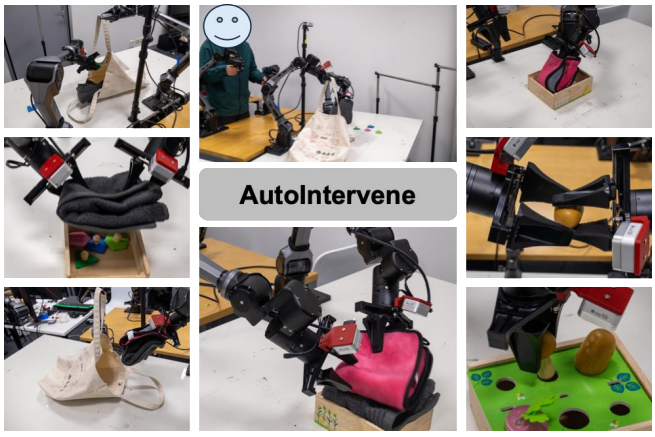

Fig. 1: AutoIntervene transfers control between the autonomous policy and the operator, thereby guiding the collection of targeted intervention data. We evaluate the system across diverse manipulation tasks.

Operator-controlled segments from successful rollouts are retained for subsequent policy updates, yielding a selective DAgger-style loop [13], [14]. Unlike standard DAgger, which queries expert actions at learner-visited states throughout a rollout [6], AutoIntervene requests supervision only during automatically identified unsupported periods. This connects deployment-time monitoring with targeted data aggregation. Figure 1 illustrates representative executions, including manipulation of deformable materials such as towels and fabric tote bags.

The contributions of this work are threefold:

1) We introduce a bidirectional intervention framework for action-chunking imitation policies, using phase-local support for policy-to-operator transfer and global support for the return to policy control.
2) We develop a mode-specific visual-action calibration procedure that uses held-out successful expert demonstrations to recompute separate intervention and recovery thresholds at each adaptation round under their

respective retrieval scopes.

3) We evaluate AutoIntervene on nine real-world tasks and show that targeted intervention trajectories support iterative policy improvement with substantially less operator-control time than collecting additional full demonstrations.

II. Related Work

Visuomotor imitation policies. Visuomotor imitation policies differ in how they represent temporally extended actions. Action chunking predicts short action sequences directly, whereas diffusion and flow-matching policies generate them through iterative or continuous-time processes [5], [15], [16]. Streaming flow trajectories can also be constrained post-training [17]. Although these approaches improve action generation, they do not by themselves determine when a deployed proposal has left demonstrated support or when control should return after intervention. AutoIntervene addresses this deployment layer independently of the action head. We evaluate it with Action Chunking with Transformers (ACT), Diffusion Policy, and Flow Matching heads.

Interactive imitation learning. Interactive policy adaptation traditionally relies on an operator to monitor policy execution and manually decide when to take over and return control. Robot-gated methods instead request intervention automatically from deployment-time signals. LazyDAgger uses policy–expert action discrepancy to trigger expert involvement, whereas RND-DAgger uses state novelty estimated by random network distillation [18], [19]. AutoIntervene follows this robot-gated direction but treats control transfer as a bidirectional, support-based process: phase-local support governs policy-to-operator transfer, global support governs operator-to-policy return, and retained recovery segments provide corrective data for subsequent policy adaptation.

Runtime policy monitoring. Runtime monitors can be compared by the signal they inspect and the action taken after detection. Error-Aware Imitation Learning detected potential failures from teleoperation data [20] and ConditionNET learned action-conditioned preconditions and effects [21]. Sentinel combined temporal action inconsistency with vision-language progress checks [8], whereas FAIL-Detect estimated failure uncertainty from successful data alone [22]. Rewind-IL further coupled calibrated inter-chunk discrepancy with respawning at a semantically verified safe state [9]. These methods provide failure signals or recovery actions, but the subsequent return of control and reuse of operator recovery remain separate system decisions. AutoIntervene instead applies the same low-level visual-action support principle to both switching directions and converts operator-controlled recovery into corrective training data.

III. AutoIntervene

AutoIntervene is designed to improve a deployed action-chunking visuomotor policy by converting automatically triggered operator interventions into targeted supervision. As illustrated in Figure 2, it consists of two stages. The first stage, **Online Control Loop**, monitors the deployed policy, transfers control to the operator when intervention is required, returns control to the policy when its proposed actions regain sufficient support, and records the resulting intervention segments. The loop proceeds sequentially through three modules: **(A) Visual-Action Query Construction**, **(B) Visual-Action Support Evaluation**, and **(C) Bidirectional Control Authority Selection**. Together, these modules construct a visual-action query from the current visual observations and policy-proposed action chunk, map it to visual-support and action-risk scores, and select control authority using their calibrated acceptance criteria. The second stage, **Policy Adaptation**, uses the retained intervention segments to update the current policy. The updated policy is subsequently redeployed, and the online control loop collects new intervention segments for the next adaptation round.

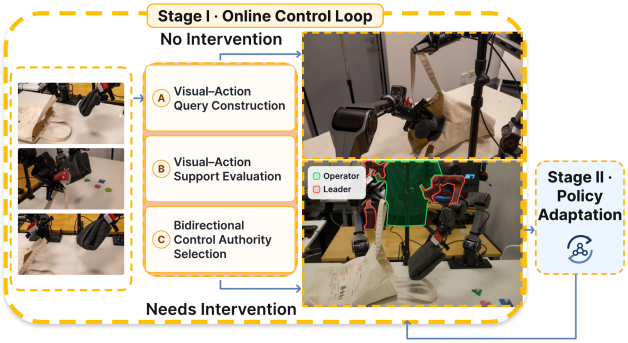

Fig. 2: Overview of *AutoIntervene*. During Stage I, the online control loop sequentially (A) constructs a visual-action query from the current visual observations and policy-proposed action chunk, (B) evaluates visual-support and action-risk scores, and (C) selects control authority using calibrated acceptance criteria. The retained intervention segments are subsequently used to adapt the policy in Stage II, after which the updated policy is redeployed.

In our leader–follower interface, the selected control authority determines the command source for the follower arms: under policy control, the follower arms execute policy-generated joint-and-gripper commands, while the leader arms track the follower configuration to remain aligned for takeover. Under operator control, the follower arms instead track the operator-manipulated leader arms.

A. Visual-Action Query Construction

During deployment, visually similar observations may occur at task phases that require different actions [8], [11]. A visual representation alone therefore does not specify what the policy intends to execute. AutoIntervene therefore considers the current visual observation together with the action chunk that the policy is about to execute. At evaluation time t, the current policy π takes the current camera images as input. Its visual encoder maps these images to embeddings $E_t = (e_t^{(1)}, \ldots, e_t^{(C)})$, where C denotes the number of camera views. The policy's action head predicts an H-step action chunk, and the monitor uses its first H_r steps for evaluation, where $1 \leq H_r \leq H$. We denote this predicted prefix by A_t.

For our bimanual system, we group the commands in A_t by arm: one group contains the left-arm joint-and-gripper commands, and the other contains the corresponding right-arm commands. Together, the visual embeddings and predicted action prefix form the visual-action query $\mathcal{Q}_t = (E_t, A_t)$.

B. Visual-Action Support Evaluation

Given the visual-action query $\mathcal{Q}_t$, AutoIntervene uses a visual-action memory $\mathcal{M}$ constructed from behaviour references and summarises their high-dimensional comparison into two scalar quantities: a visual-support score and an action-risk score.

Visual-action memory and mode-specific retrieval. The visual-action memory is constructed from all trajectories used to train the current policy π. Suppose that there are N such trajectories, collected in $\mathcal{D} = \{\tau_i\}_{i=1}^{N}$, where τ_i is the i-th trajectory in the collection. Each trajectory in $\mathcal{D}$ is used to generate the corresponding visual-action memory entries.

For any trajectory $\tau_i \in \mathcal{D}$, its length is $|\tau_i|$ recorded timesteps, and it contributes $|\tau_i| - H_r + 1$ memory entries. These entries correspond to all starting timesteps from which a complete H_r-step action chunk can be extracted. For any such entry, let $u \in \{1, \ldots, |\tau_i| - H_r + 1\}$ be its starting timestep. The H_r recorded actions from timestep u through timestep $u + H_r - 1$ form the action chunk $A_{i,u} = (a_{i,u}, \ldots, a_{i,u+H_r-1})$, while $E_{i,u}$ is the multi-view visual embedding of trajectory τ_i at the same starting timestep. Together, $A_{i,u}$ and $E_{i,u}$ form the memory entry $m_{i,u} = (E_{i,u}, A_{i,u})$. Collecting the entries generated from all valid starting timesteps across all trajectories yields the complete visual-action memory $\mathcal{M}$, as illustrated in Figure 3.

Once the visual-action memory $\mathcal{M}$ has been constructed, AutoIntervene uses it at a mode-specific evaluation frequency during deployment to continually assess the current visual-action query. At each evaluation time t, either the policy or the operator controls the bimanual system, so AutoIntervene determines the memory entries used for the current assessment according to the active control mode. Let $\beta \in \{\text{pol}, \text{op}\}$ denote this mode, where $\beta = \text{pol}$ indicates that the policy controls the bimanual system and $\beta = \text{op}$ indicates that the operator controls it. The corresponding retrieval set is denoted by $\mathcal{R}_{\beta,t} \subseteq \mathcal{M}$.

When $\beta = \text{op}$, the operator controls the bimanual system while the policy continues to generate predictions in the background. The operator's recovery may change the object state or complete part of the task, so the current state need not remain near the task phase at which the intervention began. To allow the policy to regain support from any valid task phase reached after recovery, AutoIntervene retrieves from the complete visual-action memory. We refer to support evaluated under this complete-memory retrieval scope as global support.

When $\beta = \text{pol}$, the policy controls the bimanual system and task progress is locally continuous across successive evaluations. However, memory entries from different task phases may contain similar visual embeddings but different

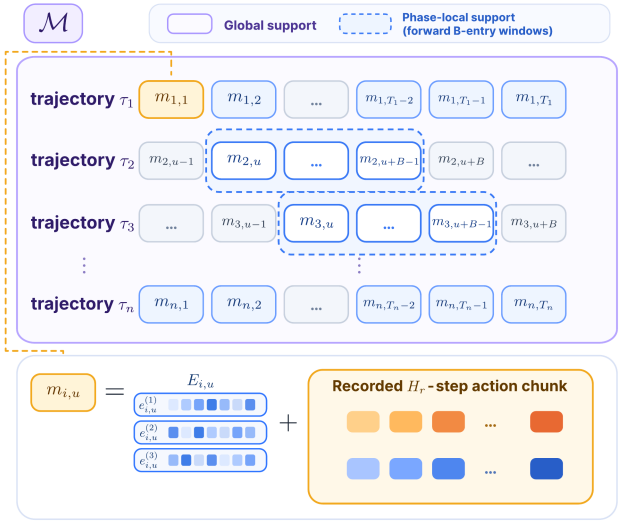

Fig. 3: Memory construction and mode-specific retrieval. $\mathcal{R}_{\text{op},t}$ uses all entries, whereas $\mathcal{R}_{\text{pol},t}$ combines forward B-entry windows from J selected trajectories.

recorded action chunks. Repeatedly searching the complete memory could therefore introduce a match from the wrong phase [11]. Whenever the policy begins controlling the bimanual system, its first evaluation compares the current visual embedding E_t with the visual embedding $E_{i,u}$ contained in every memory entry $m_{i,u} \in \mathcal{M}$. AutoIntervene retains the best-matching memory entry on each training trajectory τ_i and then selects the J trajectories whose retained entries have the strongest matches. Each selected trajectory contributes a forward window containing its matched entry and at most the next $B - 1$ entries. Subsequent evaluations retrieve only within these windows. After every U_{win} successfully executed policy actions, each window advances along its trajectory according to the newly matched entry without returning to an earlier task phase. Let $\mathcal{W}_{j,t}$ denote the window from the j-th selected trajectory at evaluation time t, where $j = 1, \ldots, J$. Thus, the mode-specific retrieval set is defined as below. We refer to support evaluated within these forward windows as phase-local support.

$$\mathcal{R}_{\beta,t} = \begin{cases} \mathcal{M}, & \beta = \text{op}, \\ \mathcal{W}_{1,t} \cup \cdots \cup \mathcal{W}_{J,t}, & \beta = \text{pol}. \end{cases}$$

For notational simplicity, throughout visual-neighbour retrieval and action-risk scoring, we write a generic memory entry as $m = (E, A)$ and suppress its trajectory and timestep indices.

Visual-neighbour retrieval. For a reference entry $m \in \mathcal{R}_{\beta,t}$, we take its multi-view visual component $E = (e^{(1)}, \ldots, e^{(C)})$ and compare it view by view with the current query embedding $E_t = (e_t^{(1)}, \ldots, e_t^{(C)})$. Their minimum cross-view visual similarity is

$$s_t(m) = \min_{1 \le c \le C} \text{sim}\left(e_t^{(c)}, e^{(c)}\right), \tag{1}$$

where sim is cosine similarity between ℓ_2-normalised embeddings. Taking the minimum prevents a high similarity in

one view from masking a mismatch in another, so every view must support the match. From $\mathcal{R}_{\beta,t}$, AutoIntervene retains up to K entries with the largest similarities as $\mathcal{N}_{\beta,t}$.

Action-risk scoring. Given $\mathcal{N}_{\beta,t}$, we compare the proposed action chunk A_t with the stored action component A of each entry $m \in \mathcal{N}_{\beta,t}$. The proposal is the chunk awaiting execution under policy control and the background policy prediction under operator control. Following the action grouping defined above, the normalised distance between these chunks for each group g is given below, where the superscript (g) selects that group's action dimensions:

$$d_{t,g}(m) = \left\| \frac{A_t^{(g)} - A^{(g)}}{\sigma_g} \right\|_2, \tag{2}$$

where σ_g contains the per-dimension standard deviations of group g's recorded actions in $\mathcal{M}$, and division is applied element-wise. The Euclidean distance is computed over all retained steps and action dimensions.

For each action group g, let the action-selected set $\mathcal{P}_g$ contain the M entries in $\mathcal{N}_{\beta,t}$ with the smallest distances. Aggregating these references reduces sensitivity to a accidental match while limiting interference from less action-relevant visual neighbours. Let $g^\star$ be the action group whose selected entries have the largest mean distance, so that risk is governed by the least-supported group rather than masked by better-matched groups. Using the same references, the mode-dependent action risk and visual support are

$$\begin{aligned} r_{\beta,t} &= \frac{1}{M} \sum_{m \in \mathcal{P}_{g^\star}} d_{t,g^\star}(m), \\ s_{\beta,t} &= \frac{1}{M} \sum_{m \in \mathcal{P}_{g^\star}} s_t(m). \end{aligned} \tag{3}$$

Both scores therefore use the same action-selected references.

Finally, $\bar{r}_{\beta,t}$ averages up to the W most recent action-risk values in the current control interval, using all available values before W evaluations. Visual support remains instantaneous so that earlier matches do not mask a sudden loss of visual correspondence. The stage returns the score pair $\left(s_{\beta,t}, \bar{r}_{\beta,t}\right)$ and, under policy control, the selected references $\mathcal{P}_{g^\star}$ for retrieval-window updates.

Policy-side retrieval-window update. After every U_{win} successfully executed policy actions, the selected references $\mathcal{P}_{g^\star}$ update the phase-local retrieval windows. For the j-th selected trajectory τ_{i_j}, let $u_{j,t}$ denote its current window position. We update this position to the largest value among its current position and the starting-timestep indices of the selected references from the same trajectory:

$$u_{j,t+1} = \max\left(\{u_{j,t}\} \cup \{u \mid m_{i_j,u} \in \mathcal{P}_{g^\star}\}\right). \tag{4}$$

C. Bidirectional Control Authority Selection

The control-authority selector receives the score pair $\left(s_{\beta,t}, \bar{r}_{\beta,t}\right)$ for the current control mode $\beta \in \{\text{pol}, \text{op}\}$ and decides whether to retain or switch control after evaluation t. This decision requires separate visual-support and action-risk thresholds for the two control modes.

Let $\mathcal{D}_{\text{cal}}$ be a fixed held-out set of successful expert trajectories excluded from policy training and the visual-action memory. Before deploying the policy in each adaptation round, we apply the same support evaluation to $\mathcal{D}_{\text{cal}}$ separately under policy and operator control. This produces the visual-support collection S_{cal}^{β} and action-risk collection R_{cal}^{β} for mode β.

Let α_s^β and α_r^β denote the prescribed lower- and upper-tail rates, respectively. With $\widehat{Q}_p$ denoting the empirical p-quantile, the corresponding thresholds are

$$\theta_s^\beta = \widehat{Q}_{\alpha_s^\beta}\left(S_{\text{cal}}^{\beta}\right), \qquad \theta_r^\beta = \widehat{Q}_{1-\alpha_r^\beta}\left(R_{\text{cal}}^{\beta}\right). \tag{5}$$

Thus, θ_s^β is the lower visual-support threshold and θ_r^β is the upper action-risk threshold. The two modes are calibrated separately because their retrieval scopes produce different score distributions.

During deployment, a policy proposal is accepted when $s_{\beta,t} \geq \theta_s^\beta$ and $\bar{r}_{\beta,t} \leq \theta_r^\beta$, and is rejected otherwise. Under policy control, a rejection, an empty phase-local reference set, or all phase-local retrieval windows reaching their final valid chunks increments the policy-side rejection counter $c_{\text{pol},t}$. Acceptance resets it. Under operator control, the policy is evaluated in the background and $c_{\text{op},t}$ counts consecutive acceptances. A rejected proposal resets it. Given the required persistence lengths L_{pol} and L_{op}, the authority update β^+ is

$$\beta^+ = \begin{cases} \text{op}, & \beta = \text{pol and } c_{\text{pol},t} \geq L_{\text{pol}}, \\ \text{pol}, & \beta = \text{op and } c_{\text{op},t} \geq L_{\text{op}}, \\ \beta, & \text{otherwise.} \end{cases} \tag{6}$$

The persistence lengths L_{pol} and L_{op} require the corresponding decision to remain consistent across successive evaluations, preventing brief score fluctuations from causing unnecessary control changes.

D. Policy Adaptation

In adaptation round k, the Online Control Loop deploys policy $\pi^{(k)}$, trained on the accumulated trajectory collection $\mathcal{D}^{(k)}$. Each interval during which the operator controls the system is retained as an intervention segment and stored as a separate intervention trajectory. The segment begins when control switches from the policy to the operator and ends when control returns to the policy or the episode terminates. The intervention trajectories retained from successful rollouts form $\Delta\mathcal{D}^{(k)}$. Keeping them separate prevents training examples from crossing a change in control source and focuses supervision on states that required operator correction [6], [13].

At the end of round k, the new intervention data are added to the current training set. For any trajectory collection $\mathcal{D}$, let $p_{\mathcal{D}}$ denote uniform sampling over its valid H-step training examples. The next training set and adaptation sampling

distribution are

$$\mathcal{D}^{(k+1)} = \mathcal{D}^{(k)} \cup \Delta\mathcal{D}^{(k)},$$
$$p^{(k+1)} = \lambda_{\text{mix}} p_{\mathcal{D}^{(k)}} + (1-\lambda_{\text{mix}}) p_{\Delta\mathcal{D}^{(k)}}, \quad 0 \le \lambda_{\text{mix}} \le 1, \quad (7)$$

This mixture retains prior training data while assigning the new intervention data a fixed sampling weight [23], [24].

The next policy $\pi^{(k+1)}$ is trained by standard behaviour cloning on examples sampled from $p^{(k+1)}$. The visual-action memory is then rebuilt from $\mathcal{D}^{(k+1)}$ using the visual encoder of $\pi^{(k+1)}$. Before redeployment, the mode-specific thresholds are recalibrated on $\mathcal{D}_{\text{cal}}$, after which $\pi^{(k+1)}$ enters the next Online Control Loop. Repeating this cycle progressively incorporates intervention data from unsupported task regions encountered during deployment.

IV. Empirical Evaluation

We evaluate *AutoIntervene* on nine real-world bimanual manipulation tasks, comprising a seven-task main benchmark and two longer-horizon extensions. The experiments are organized around four questions: **Q1:** Does targeted intervention improve policy success relative to the initial policy and additional full demonstrations? **Q2:** Does automatic switching achieve a better success–control-data tradeoff than manual switching and collecting additional full demonstrations? **Q3:** Can *AutoIntervene* support iterative adaptation on long-horizon tasks and remain compatible with different action-generation heads? **Q4:** How reliably does *AutoIntervene* perform bidirectional handoff relative to prior monitors, and which components contribute to policy-to-operator and operator-to-policy switching?

Experimental setup. All experiments used two AgileX PiPER-X 6-DoF arms with stock parallel grippers and three RGB cameras: one overhead and one on each wrist. Policy observations comprised a 28-dimensional bimanual state, including 14 joint-and-gripper values and their corresponding 14 measured-torque values, while actions were 14-dimensional joint-and-gripper commands. For the Diffusion Policy and Flow Matching variants, we retained the same robot, observation, and deployment interfaces while replacing only the action-generation head. Demonstrations and interventions used ALOHA-style leader-follower teleoperation with TriPilot-FF-style arm-side force reflection [5], [25]. Our fixed-base setup omitted its pedal and mobile-base channels. We instantiated the policy's visual encoder with DINOv3 ConvNeXt-Base [26]. We used two action groups corresponding to the left and right arms.

Data and evaluation protocol. For each task, 30 of the 36 initial trajectories formed $\mathcal{D}^{(0)}$ and the remaining six formed the fixed $\mathcal{D}_{\text{cal}}$. Subsequent intervention trajectories augmented both the training set and visual-action memory but not the fixed calibration set. For both Human and AutoIntervene, each adaptation round retained intervention trajectories from five successful deployment rollouts per task. Unless stated otherwise, every task-method success rate was measured over 25 unassisted physical rollouts.

TABLE I: Definitions of the evaluated tasks.

Task	Description
Peg Disassembly	Disassemble a pegged object and place all parts into the box.
Potato Transfer	Transfer a wooden potato between hands and place it into the box.
Towel Folding	Fold a towel from a fixed initial configuration.
Towel Bagging	Fold a towel, pack it and the pegs into a bag, and move the bag to the target side.
Lidded Box Packing	Open the box, pack the objects inside, and close the lid.
Plant Sorting	Place three wooden plant objects into their matching box slots.
Towel Box Packing	Remove pegs, fold the towel, reposition the box, and place the towel inside.
Two-Towel Box Packing	Remove pegs, fold two towels, and place both into the box.
Towels-and-Cable Bagging	Fold two towels, pick up the cable, and place all three items into the bag.

Success required completing every subtask in Table I. The main benchmark comprised seven bimanual manipulation tasks, while two longer-horizon extensions evaluated iterative adaptation. Table I defines all nine tasks. Figures 4 and 6 show representative executions.

Implementation details. Policies used 256×256 images and an action-chunk horizon of $H = 100$ future actions. All policy variants used the same encoder–decoder backbone, consisting of one encoder layer and four decoder layers, while ACT, Diffusion Policy, and Flow Matching differed only in their action-generation mechanisms. During physical deployment, policy inference and visual-action monitoring ran on single NVIDIA RTX 5090. For adaptation, accumulated pre-round training data and newly collected round-k intervention data were sampled in a 2:1 ratio ($\lambda_{\text{mix}} = 2/3$). The monitor used $J = K = 16$, $B = H_r = 40$, $M = 3$, and $W = 5$. After a 1 s warm-up, under policy control, policy actions were executed at 30 Hz, whereas AutoIntervene evaluation was performed at 5 Hz. The phase-local retrieval windows were therefore updated every $U_{\text{win}} = 6$ successfully executed policy actions. Under operator control, teleoperation commands were executed at 200 Hz, whereas AutoIntervene evaluation was performed at 30 Hz. Both directions required two consecutive decisions ($L_{\text{pol}} = L_{\text{op}} = 2$). Each round recomputed its thresholds with $(\alpha_s^{\text{pol}}, \alpha_r^{\text{pol}}) = (0.05, 0.05)$ and $(\alpha_s^{\text{op}}, \alpha_r^{\text{op}}) = (0.30, 0.30)$.

Baseline monitor settings. For the controlled comparison, we implemented deployment-time handoff monitors adapted from LazyDAgger [18] and RND-DAgger [19] to the ACT deployment setting, while following their original switching criteria and reported threshold-setting procedures. For RND-DAgger, we followed its reported threshold rule, set the threshold to twice the mean training-set RND score, and tested $W_{\text{rec}} \in \{5, 30\}$.

Compared methods. The main benchmark compares four settings. **Initial** trains only on the 30 original training demonstrations. **Additional Full Data** adds ten full expert trajectories per task from the nominal initial-state distribution. **Human** uses the same intervention-learning pipeline

TABLE II: Success (%) and recorded control-data time across seven tasks. Human and AutoIntervene use manually and automatically triggered interventions, respectively. R1 and R2 denote adaptation rounds. Δt is cumulative additional operator-control time, while Initial and Additional Full Data report total demonstration time.

Task	Initial		Human R1		Human R2		AUTOINTERVENE R1		AUTOINTERVENE R2		Additional Full Data	
	Succ. (%)	Time (s)	Succ. (%)	Δt (s)	Succ. (%)	Δt (s)	Succ. (%)	Δt (s)	Succ. (%)	Δt (s)	Succ. (%)	Time (s)
Peg Disassembly	16%	1173.7	52%	123.6	56%	286.9	64%	49.3	**72%**	127.1	60%	1853.0
Potato Transfer	44%	368.7	52%	20.7	**80%**	74.2	48%	29.6	76%	50.2	64%	547.3
Towel Folding	56%	766.2	60%	36.4	**100%**	144.8	76%	61.4	96%	140.4	68%	1100.0
Towel Bagging	24%	1761.1	32%	130.2	36%	266.5	44%	121.6	**60%**	171.0	40%	2536.7
Lidded Box Packing	8%	989.6	20%	107.5	92%	246.7	52%	63.1	**100%**	130.2	60%	1440.0
Plant Sorting	24%	450.2	52%	88.8	56%	157.1	52%	76.6	**72%**	128.9	32%	693.1
Towel Box Packing	44%	1332.4	52%	36.4	60%	83.3	80%	61.4	**84%**	112.3	68%	1930.5
Average	30.9%	977.4	45.7%	77.7	68.6%	179.9	59.4%	66.1	**80.0%**	122.9	56.0%	1442.9

Fig. 4: Policies adapted with AutoIntervene handle deformable objects in the Towels-and-Cable Bagging task, including cloth, cables, and a tote bag.

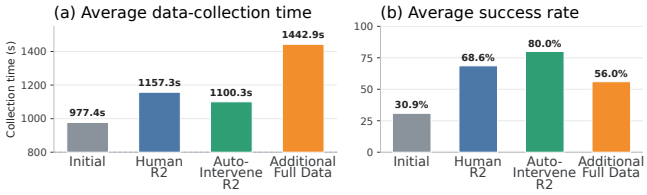

Fig. 5: Across-task mean success and total recorded control-data time after R2, including the shared initial demonstrations.

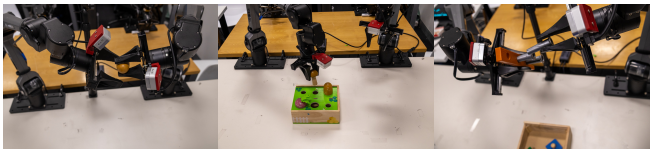

Fig. 6: Policies adapted with AutoIntervene perform precise manipulation in the Plant Sorting task, including insertion and handovers.

as our method, but an operator manually selects policy-to-operator and operator-to-policy switches. Its retained intervention segments are mixed with previous training data as separate intervention trajectories. **AUTOINTERVENE** uses the proposed visual-action support evaluation to trigger both switching directions automatically. For both intervention-based methods, R1 and R2 denote successive adaptation rounds.

Metrics. The primary metric is task success rate. To measure data efficiency, we also report recorded control-data time. Table II reports demonstration time for Initial and Additional Full Data and additional operator-control time for Human and AUTOINTERVENE. For all methods, Figure 5 reports total time by adding the shared initial demonstration time to the intervention methods.

A. Targeted Intervention Improves Policy Adaptation (Q1)

As shown in Table II, across the seven-task benchmark, targeted intervention consistently improves policy adaptation: after two rounds, both Human and AUTOINTERVENE outperform the Initial policy and Additional Full Data in aggregate. AUTOINTERVENE delivers the strongest improvement, increasing the across-task mean success rate by 49.1 percentage points, with gains across all seven tasks. These results support AUTOINTERVENE as an effective approach for collecting targeted corrective supervision from learner-induced failure states.

B. Automatic Switching Enables More Efficient Policy Adaptation (Q2)

As shown in Table II and Figure 5, AUTOINTERVENE achieves higher average success than both Human and Additional Full Data, while using approximately 74% less additional recorded control-data time than Additional Full Data. Relative to manual switching, AUTOINTERVENE returns control to the policy once the recovered state and policy action return to demonstrated support, keeping each retained intervention segment focused on the failure and recovery. A manually selected operator-to-policy switch depends on operator judgment and button timing, so the segment can extend beyond recovery into states that the policy already handles. Training on this extra tail dilutes the corrective update by adding supervision in already-supported regions. Additional Full Data is also less targeted because it spends collection time on complete nominal trajectories rather than learner-induced failure states.

TABLE III: Success (%) over three AutoIntervene adaptation rounds on two long-horizon tasks.

Training Stage	Two-Towel Box Packing	Towels-and-Cable Bagging
Initial Policy	28%	8%
AutoIntervene R1	44%	20%
AutoIntervene R2	64%	28%
AutoIntervene R3	**88%**	**48%**
Additional Full Data	52%	28%

TABLE IV: Success (%) over three AutoIntervene adaptation rounds on Two-Towel Box Packing with Diffusion Policy (DP), Flow Matching (FM), and ACT action heads.

Training Stage	DP	FM	ACT
Initial Policy	32%	32%	28%
AutoIntervene R1	40%	36%	44%
AutoIntervene R2	56%	48%	64%
AutoIntervene R3	**92%**	**80%**	**88%**
Additional Full Data	44%	40%	52%

C. Iterative Adaptation for Longer-Horizon Tasks (Q3)

To evaluate *AutoIntervene* on longer task executions, we perform three adaptation rounds on two longer-horizon manipulation tasks. As shown in Table III, success improves after each round on both tasks, and the final policies outperform those trained with Additional Full Data. These results show that *AutoIntervene* is not limited to correcting a single local failure: by collecting intervention data from the problems encountered during each deployment and using it to update the policy, it continues improving performance over longer, more complex task executions.

D. Adaptation across ACT, Diffusion, and Flow-Matching Heads (Q3)

To test whether *AutoIntervene* transfers beyond ACT, we replace the ACT action head with Diffusion Policy and Flow Matching heads while keeping the shared backbone, robot interfaces, and monitor unchanged. As shown in Table IV, all three action heads improve over successive adaptation rounds, and their R3 policies outperform both the Initial policies and Additional Full Data. *AutoIntervene* therefore transfers across different action-generation mechanisms without head-specific modification.

E. AutoIntervene Outperforms Prior Handoff Monitors through Complementary Components (Q4)

Protocol and metrics. Handoff is evaluated during lid opening in Lidded Box Packing using the same 20,000-step policy checkpoint, physical configuration, and pre-specified monitor parameters. Each method is tested in 10 rollouts after a fixed 5 cm box translation and 10 nominal rollouts, with completion, 60 s, or five handoff cycles terminating a rollout. The perturbed condition tests whether the monitor transfers control to the operator after a policy failure and returns it to the policy after operator recovery. The nominal condition tests whether a successful autonomous execution remains uninterrupted when no intervention is needed. A valid **cut-in** occurs between displacement and 3 s after the resulting failed grasp. A valid **cut-out** is the operator-to-policy switch after the operator restores the box and guides the gripper to the lid. For the four transition metrics on perturbed rollouts, let V_d, M_d, and E_d denote the numbers of valid transitions, missed opportunities, and extra transitions for $d \in \{\text{cut-in}, \text{cut-out}\}$. We compute

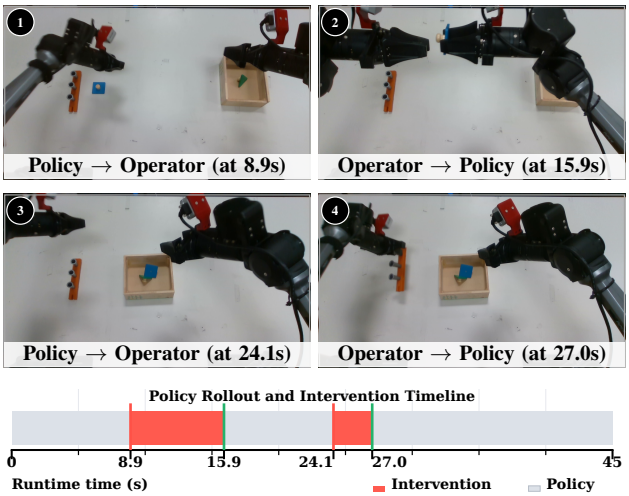

Fig. 7: One bidirectional intervention episode on Peg Disassembly. Top: two policy-to-operator switches and their corresponding operator-to-policy switches. Bottom: the same timeline, with policy control in grey and operator control in red.

$$\text{Recall}_d = \frac{V_d}{V_d + M_d}, \qquad \text{Precision}_d = \frac{V_d}{V_d + E_d}. \tag{8}$$

For these four metrics, a zero denominator is reported as N/A. A valid cut-out must follow a valid cut-in. Consequently, cut-out recall is conditioned on valid cut-ins, whereas cut-out precision considers all attempted operator-to-policy switches. The false-trigger rate is the fraction of nominal rollouts containing at least one unnecessary cut-in, with each rollout counted at most once regardless of repeated triggers.

The comparison includes LazyDAgger [18] and RND-DAgger [19], with the two RND-DAgger settings differing only in recovery persistence W_{rec}. The ablations preserve retrieval and neighbour selection, removing only visual support or action risk from the acceptance criteria.

Comparison with prior monitors. As shown in Table V, reliable bidirectional handoff without unnecessary policy-to-operator switches during nominal execution is achieved only by AutoIntervene. LazyDAgger misses failures and retains operator control because policy–operator disagreement remains large during recovery. RND-DAgger uses one novelty threshold for both directions. After an operator-to-policy switch, a score increase can cross the same threshold and immediately return control to the operator. Increasing recovery persistence only delays the return to policy control, whereas AutoIntervene uses separately calibrated, mode-specific switching criteria to implement hysteretic switching and suppress immediate reversal.

Ablations of visual support and action risk. As shown

TABLE V: Controlled handoff comparison on Lidded Box Packing using 10 perturbed and 10 nominal rollouts per method, capped at five complete handoff cycles. Values are rates. N/A indicates an undefined denominator.

	Perturbed				Nominal
Method	Cut-in Recall ↑	Cut-in Prec. ↑	Cut-out Recall ↑	Cut-out Prec. ↑	False-trigger Rate ↓
AutoIntervene	**1.00**	**1.00**	**1.00**	**1.00**	**0.00**
Prior handoff monitors					
LazyDAgger [18]	0.40	**1.00**	0.00	N/A	0.80
RND-DAgger ($W_{\mathrm{rec}} = 5$) [19]	0.80	0.16	0.00	0.00	0.90
RND-DAgger ($W_{\mathrm{rec}} = 30$) [19]	0.80	0.16	0.75	0.12	0.90
Ablations of visual support and action risk					
w/o visual support	0.00	N/A	N/A	N/A	**0.00**
w/o action risk	0.90	**1.00**	0.56	0.56	**0.00**

in Table V, without visual support, *AutoIntervene* misses the workspace displacement because action risk can remain low despite weak visual correspondence. Without action risk, most policy-to-operator switches remain, but the return to policy control may be less reliable when visual correspondence recovers before the proposal matches successful reference actions. Thus, visual support detects when intervention is needed and action risk prevents a premature operator-to-policy switch. Both are required for reliable bidirectional handoff.

Ablation of policy-side retrieval-window updates. Beyond the score components, we evaluate policy-side retrieval-window updates by lowering the target object while preserving a similar RGB appearance, causing the nominal grasp to fail. Across 10 rollouts per setting, *AutoIntervene* with retrieval-window updates achieves a cut-in recall of 1.00. During repeated-grasp failures, the updates advance the phase-local windows until they all reach their final valid chunks, incrementing the policy-side rejection counter until control transfers to the operator. Without these updates, the policy can repeat failed grasps without transferring control, yielding a cut-in recall of 0.30.

V. Conclusion

AutoIntervene combines phase-local visual-action monitoring, bidirectional handoff, and intervention-data aggregation. Across nine real-world bimanual tasks, it improved policies over successive rounds with less additional control data than full demonstrations, achieved higher mean success than manual switching, and required less operator-control time on average. Together with separately calibrated criteria for both switching directions, these components focus operator control on unsupported periods and resume autonomy once the policy proposal returns to demonstrated support. Future work will extend online calibration across broader tasks, perturbations, and operators.

References

[1] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Foundations and Trends in Robotics*, 2018.

[2] W. Zhi, T. Lai, L. Ott, and F. Ramos, "Diffeomorphic transforms for generalised imitation learning," in *Proceedings of the 4th Annual Learning for Dynamics and Control Conference*, ser. Proceedings of Machine Learning Research, vol. 168, 2022, pp. 508–519.

[3] A. Mandlekar *et al.*, "What matters in learning from offline human demonstrations for robot manipulation," in *Proceedings of the 5th Conference on Robot Learning*, 2022.

[4] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei, "GTI: Learning to generalize across long-horizon tasks from human demonstrations," in *Robotics: Science and Systems*, 2020.

[5] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," in *Proceedings of Robotics: Science and Systems*, 2023.

[6] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011.

[7] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "DART: Noise injection for robust imitation learning," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017.

[8] C. Agia *et al.*, "Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress," in *Proceedings of the 8th Conference on Robot Learning*, 2025.

[9] G. Zheng, S. Seenivasan, M. Johnson-Roberson, and W. Zhi, "Rewind-IL: Online failure detection and state respawning for imitation learning," arXiv preprint arXiv:2604.16683, 2026.

[10] P. Sermanet *et al.*, "Time-contrastive networks: Self-supervised learning from video," in *IEEE International Conference on Robotics and Automation*, 2018.

[11] Z. Ouyang, K. Wang, J. Liu, H. Lu, and W. Zhang, "SCIL: Stage-conditioned imitation learning for multi-stage manipulation," *IEEE Control Systems Letters*, 2025.

[12] Z. Li, R. Qiu, Y. Chen, G. Ren, and W. Zhi, "TRACE: Trajectory-routed causal memory for delayed-evidence visuomotor imitation," arXiv preprint arXiv:2606.14551, 2026.

[13] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, "HG-DAgger: Interactive imitation learning with human experts," in *IEEE International Conference on Robotics and Automation*, 2019.

[14] Z. Hu *et al.*, "RaC: Robot learning for long-horizon tasks by scaling recovery and correction," arXiv preprint arXiv:2509.07953, 2025.

[15] C. Chi *et al.*, "Diffusion policy: Visuomotor policy learning via action diffusion," in *Proceedings of Robotics: Science and Systems*, 2023.

[16] Q. Zhang, Z. Liu, H. Fan, G. Liu, B. Zeng, and S. Liu, "FlowPolicy: Enabling fast and robust 3D flow-based policy via consistency flow matching for robot manipulation," in *AAAI Conference on Artificial Intelligence*, 2025.

[17] J. Long, D. Liu, W. Cai, I. Manchester, and W. Zhi, "Safe policies post-training: Constraining streaming flow models for adapting learned robot trajectory distributions," *IEEE Robotics and Automation Letters*, vol. 11, no. 9, pp. 10 656–10 663, 2026.

[18] R. Hoque *et al.*, "LazyDAgger: Reducing context switching in interactive imitation learning," in *IEEE International Conference on Automation Science and Engineering*, 2021.

[19] E. Biré, A. Kobanda, L. Denoyer, and R. Portelas, "Efficient active imitation learning with random network distillation," in *International Conference on Learning Representations*, 2025.

[20] J. Wong *et al.*, "Error-aware imitation learning from teleoperation data for mobile manipulation," in *Proceedings of the 5th Conference on Robot Learning*, 2022.

[21] D. Sliwowski and D. Lee, "ConditionNET: Learning preconditions and effects for execution monitoring," *IEEE Robotics and Automation Letters*, 2025.

[22] C. Xu *et al.*, "Can we detect failures without failure data? Uncertainty-Aware runtime failure detection for imitation learning policies," in *Proceedings of Robotics: Science and Systems*, 2025.

[23] A. Mandlekar, D. Xu, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese, "Human-in-the-loop imitation learning using remote teleoperation," *arXiv preprint arXiv:2012.06733*, 2020.

[24] D. Rolnick, A. Ahuja, J. Schwarz, T. P. Lillicrap, and G. Wayne, "Experience replay for continual learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[25] Z. Li, Y. Zhou, R. Qiu, H. Wu, G. Ren, and W. Zhi, "TriPilot-FF: Coordinated whole-body teleoperation with force feedback," arXiv preprint arXiv:2602.09888, 2026.

[26] O. Siméoni *et al.*, "DINOv3," *Transactions on Machine Learning Research*, 2026.