REAL-TIME ACTION CHUNKING FOR RELATIVE-ACTION ROBOT POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Relative-action representations offer an appealing action space for robot learning, as they reduce dependence on a specific robot configuration and facilitate scalable robot-free data collection. In such policies, each action chunk is expressed relative to its initial end-effector pose, referred to as the anchor pose. Deploying these policies on robots requires real-time execution for fast and continuous operation. Existing real-time execution methods achieve this by aligning the overlapping actions of consecutive chunks. However, for relative-action policies, such alignment does not ensure consistent absolute target poses because consecutive chunks use different anchor poses. Controller tracking error can further shift these anchors during execution, introducing additional cross-chunk inconsistency. Robot-free demonstrations make this problem more challenging, as they provide no direct supervision for deployment-time tracking errors. In this work, we introduce **RRTC**, a tracking-error-aware **R**elative-action **R**eal-**T**ime **C**hunking framework. RRTC re-expresses the overlapping trajectory of the preceding chunk relative to the robot's realized pose, thereby preserving cross-chunk absolute-target consistency. It further conditions action generation on the tracking error, allowing the policy to adapt accordingly. To enable such training with robot-free demonstration data, RRTC synthesizes pseudo tracking errors from temporal offsets along demonstration trajectories. Experiments across diverse simulated and real-world manipulation tasks show that RRTC enables smoother and more efficient real-time execution while maintaining robust task performance. Source code is available at `https://anonymous.4open.science/r/RRTC-ICLR-2141`.

1 INTRODUCTION

Recent advances in imitation learning have enabled robots to acquire complex manipulation skills from human demonstrations, while robot-free demonstration collection has emerged as an attractive approach for scaling such data. In this setting, human operators record trajectories using handheld devices rather than teleoperating physical robots (Chi et al., 2024), and the resulting actions are commonly represented in a relative-action space (Feng et al., 2026). Specifically, each action chunk is expressed with respect to the end-effector pose at the beginning of the chunk, which we refer to as the anchor pose. This representation reduces dependence on a particular robot coordinate frame and has been adopted by several recent large-scale robot-free manipulation datasets (Liu et al., 2025; 2026). Relative-action policies are therefore becoming an increasingly important paradigm for scalable robot policy learning.

Deploying these policies on physical robots, however, requires responsive real-time execution. Modern visuomotor policies, including Vision-Language-Action (VLA) models, often incur non-negligible inference latency. If the robot remains idle while waiting for each new action chunk, execution becomes slow and responses to environmental changes are delayed. Real-time execution methods therefore generate the next chunk asynchronously while the robot continues executing the current one. Because consecutive chunks are generated independently from different observations, they may predict different trajectories over their overlapping time steps, causing abrupt target pose changes and robot jitter at chunk handover (Black et al., 2025a). Existing real-time chunking methods address this problem effectively in the absolute-action setting by aligning the overlapping predictions of consecutive chunks, thereby promoting smooth handovers (Black et al., 2025a;b; Tang et al., 2025; Wang et al., 2026b).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

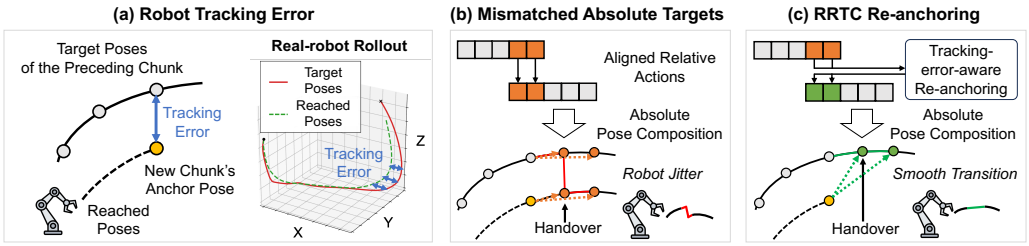


Figure 1: (a) Tracking error shifts new chunk's anchor from the target pose of the preceding chunk. (b) Aligned relative actions can correspond to mismatched absolute targets, causing robot jitter. (c) RRTC re-anchors the overlapping relative actions, restoring absolute target consistency.

However, directly applying this alignment strategy to relative-action policies leads to two new problems. First, unlike an absolute action, a relative action does not uniquely determine a physical target pose, since its target also depends on the chunk's anchor pose. Consecutive chunks are anchored at different robot poses. During execution, controller tracking error can further shift the new chunk's anchor pose because the robot may not exactly reach the target pose (Dawson et al., 1990; Klančar & Škrjanc, 2007), as shown in Figure 1 (a). As illustrated in Figure 1 (b), even when the overlapping relative actions of two chunks are aligned, they can still correspond to different absolute target poses. This mismatch introduces a discontinuity in the target poses and causes robot jitter at handover. In other words, action-space consistency does not guarantee target-space consistency for relative-action policies. Second, addressing this mismatch also introduces a training challenge. During deployment, the tracking error can be computed from the difference between the target and reached robot poses. Robot-free demonstrations, however, contain only reached demonstration trajectories and therefore provide no direct supervision for such deployment-time errors.

We address these challenges with **RRTC**, a tracking-error-aware **R**elative-action **R**eal-**T**ime **C**hunking framework built on three key ideas. First, RRTC re-anchors the overlapping actions from the preceding chunk with respect to the reached anchor of the new chunk, preserving the same absolute target poses despite tracking-induced anchor shifts, as illustrated in Figure 1 (c). The resulting re-anchored actions are then used as a fixed prefix for generating the new chunk. Second, RRTC explicitly conditions action generation on the current tracking error, enabling the generated suffix to adapt to the robot's deviation from the target trajectory. Third, since robot-free demonstrations contain no tracking-error supervision, RRTC synthesizes pseudo tracking errors from temporal offsets along demonstration trajectories and constructs the corresponding re-anchored prefixes and training targets. Together, these designs preserve cross-chunk absolute-target consistency while enabling the policy to adapt to deployment-time tracking errors, leading to smoother, faster, and more reliable real-time execution. In summary, our contributions are threefold:

- We identify and formulate a previously overlooked failure mode in real-time execution of relative-action policies: controller tracking error shifts the action anchor, such that aligning relative actions across chunks does not guarantee consistency of their absolute target poses.
- We propose **RRTC**, a tracking-error-aware real-time chunking framework that preserves cross-chunk absolute targets through re-anchoring and adapts future action generation to tracking error, using pseudo errors synthesized from demonstration trajectories for training.
- We extensively evaluate RRTC across simulated and real-world tasks under diverse execution conditions, demonstrating higher task success, faster execution, and smoother robot motion.

2 Related Work

Visuomotor Policies and Action Chunking. Recent visuomotor policies (Fu et al., 2024; Octo Model Team et al., 2024; Chi et al., 2025), including Vision-Language-Action (VLA) models (Kim et al., 2024; Physical Intelligence et al., 2025), have enabled increasingly complex robot manipulation. Flow-based policies, such as Diffusion Policy and $\pi_{0.5}$, are widely adopted in these systems for modeling multimodal action distributions (Chi et al., 2025; Physical Intelligence et al., 2025; Bjorck et al., 2025), and constitute the policy class considered in this work. These policies commonly employ action chunking (Zhao et al., 2023; Kim et al., 2025; Chi et al., 2025; Physical Intelligence et al., 2025; Bjorck et al., 2025), where the policy predicts a sequence of future actions at each inference step.

Robot-Free Demonstration Learning and Relative-Action Policies. To enable scalable data collection and cross-embodiment generalization, recent works have explored robot-free demonstration learning. The Universal Manipulation Interface (UMI) (Chi et al., 2024) develops a handheld demonstration-collection gripper. Owing to its high data-collection efficiency, this paradigm has inspired a growing number of large-scale datasets (Liu et al., 2025; 2026). Since robot absolute poses and joint states are unavailable during robot-free demonstrations, actions are typically represented in delta-action spaces (Chi et al., 2024; Octo Model Team et al., 2024; Kim et al., 2024). As analyzed by (Feng et al., 2026), among such representations, the *relative-action* representation (Chi et al., 2024), also referred to as *chunk-wise delta*, achieves better performance by avoiding the accumulation of action errors. Accordingly, we focus on policies operating in this action space.

Real-Time Execution of Action-Chunking Policies. Real-time execution frameworks allow robots to execute during policy inference (Black et al., 2025a;b; Tang et al., 2025; Arachchige et al., 2025; Wang et al., 2026b). A key challenge is maintaining consistency between chunks. RTC (Black et al., 2025a) addresses this problem through test-time inpainting, which conditions the new chunk on an action prefix from the current chunk. To eliminate the additional inference overhead, Training-Time RTC (TT-RTC) (Black et al., 2025b) trains the policy to directly condition on the action prefix. REMAC (Wang et al., 2026b) further addresses both inter-chunk and intra-chunk consistency. Despite these advances, existing methods do not explicitly account for robot tracking error during real-time execution of relative-action policies.

3 Rethinking Real-Time Relative-Action Chunking

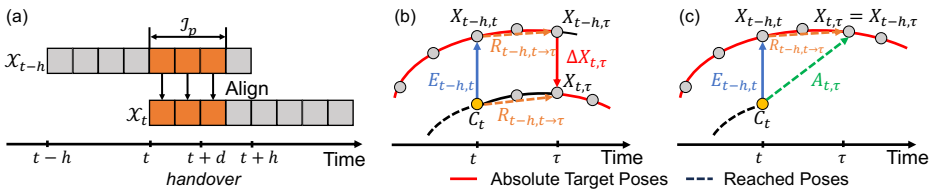


Figure 2: (a) Illustration of real-time execution of absolute-action policies. (b) Tracking error shifts the anchor pose, causing cross-chunk inconsistency when the relative action is directly aligned. (c) Tracking-error-aware relative action restores cross-chunk consistency.

3.1 Preliminaries

Real-Time Execution of Absolute-Action Policies. Consider a policy $\pi(\mathcal{X}_t \mid o_t)$ that predicts a chunk of absolute target poses $\mathcal{X}_t = \{X_{t,k}\}_{k=t}^{t+H-1}$, where H denotes the prediction horizon, o_t is the observation at time step t, and k denotes the intended execution time. Each $X_{t,k} \in SE(3)$ specifies an end-effector target pose.

Real-time execution frameworks perform policy inference while the robot continues executing. Let d denote the inference delay and h the *execution horizon*, such that a new action chunk is generated every h steps. Specifically, while the robot is executing $\mathcal{X}_{t-h}$, the policy starts generating the new chunk $\mathcal{X}_t$ at time step t and returns it at $t+d$. The two chunks overlap over $\mathcal{I} = \{t, \dots, t-h+H-1\}$. Once $\mathcal{X}_t$ becomes available at $t+d$, execution switches from the preceding chunk to the new chunk. Since $\mathcal{X}_t$ and $\mathcal{X}_{t-h}$ are generated independently, their target poses may be inconsistent over the overlapping interval $\mathcal{I}$. At the handover step $t+d \in \mathcal{I}$, let $\Delta X_{t+d} = X_{t-h,t+d}^{-1} X_{t,t+d} \in SE(3)$ denote the chunk shift. It leads to a discontinuity when switching between chunks, causing robot jitter and reducing execution efficiency. Existing real-time execution frameworks (Black et al., 2025a;b; Wang et al., 2026b) seek to eliminate this discontinuity by aligning the overlapping target poses within a prefix window $\mathcal{I}_p = \{t, \dots, t+L_p-1\}$ through inference- or training-time conditioning. Here, L_p denotes the prefix horizon, and satisfies $d \le L_p \le h$. Specifically, they align $X_{t,\tau} \approx X_{t-h,\tau}$, as shown in Figure 2 (a), such that $\Delta X_\tau \approx I$ for $\tau \in \mathcal{I}_p$.

Relative-Action Chunking. Unlike the absolute-pose formulation above, a relative-action chunking policy $\pi(\mathcal{A}_t \mid o_t)$ predicts a chunk of relative pose actions $\mathcal{A}_t = \{A_{t,k}\}_{k=t}^{t+H-1}$. Each relative action $A_{t,k} \in SE(3)$ specifies an end-effector pose target relative to the anchor pose $C_t \in SE(3)$, given by the observed end-effector pose at time step t (Chi et al., 2024). For robot control, each relative action is composed with its anchor pose to obtain the corresponding absolute target pose $X_{t,k} = C_t A_{t,k}$ (Wang et al., 2026a).

Tracking Error. During physical robot execution, the reached end-effector pose may deviate from the target pose due to finite controller bandwidth, robot dynamics, and execution disturbances (Chi et al., 2024; Arachchige et al., 2025; Zhao et al., 2026). Such deviations generally persist and cannot be completely eliminated, especially during continuous and high-speed execution. Let $C_k \in SE(3)$ denote the reached end-effector pose at time step k. For an absolute target pose $X_{t,k}$ prescribed by the policy from observation o_t, the corresponding tracking error is expressed as $E_{t,k} = C_k^{-1} X_{t,k} \in SE(3)$, where $E_{t,k} = I$ indicates perfect tracking.

3.2 Tracking Error in Real-Time Execution of Relative-Action Policies

For policies that directly predict absolute target poses, existing real-time execution methods preserve smooth chunk transitions by aligning the overlapping absolute target poses, i.e., $X_{t,\tau} \approx X_{t-h,\tau}$ for $\tau \in \mathcal{I}_p$. For relative-action policies, however, the new absolute target is given by $X_{t,\tau} = C_t A_{t,\tau}$. Therefore, maintaining the same cross-chunk target consistency requires $A_{t,\tau} \approx C_t^{-1} X_{t-h,\tau}$.

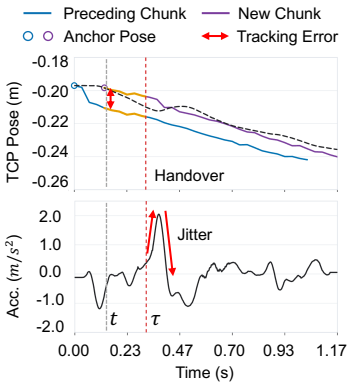


Figure 3: Tracking error shifts the anchor pose, causing jitter.

To interpret this, consider the absolute pose planned by the preceding chunk at time step t. The preceding chunk expects the robot to reach $X_{t-h,t}$, while the reached pose is C_t, which serves as the anchor of the new chunk. Their discrepancy is the tracking error $E_{t-h,t} = C_t^{-1} X_{t-h,t}$. We further denote the remaining relative actions planned by the preceding chunk from time t to an overlapping time step τ as $R_{t-h,t\to\tau} = X_{t-h,t}^{-1} X_{t-h,\tau}$. Substituting these terms into the consistency condition gives $A_{t,\tau} \approx E_{t-h,t} R_{t-h,t\to\tau}$, as shown in Figure 2 (c). This reveals that the relative action required for cross-chunk consistency depends directly on the tracking error. Under perfect tracking, $E_{t-h,t} = I$, and the required action reduces to the remaining motion $R_{t-h,t\to\tau}$. When tracking error exists, however, using the remaining motion alone yields $X_{t,\tau} = C_t R_{t-h,t\to\tau}$ instead of the intended target $X_{t-h,\tau} = X_{t-h,t} R_{t-h,t\to\tau}$. Their target discrepancy is $\Delta X_{t,\tau} = X_{t-h,\tau}^{-1} X_{t,\tau} = R_{t-h,t\to\tau}^{-1} E_{t-h,t}^{-1} R_{t-h,t\to\tau}$, showing that tracking error induces a discontinuity at chunk handover, as shown in Figure 2 (b). This effect is observed on a real robot, where the tracking-error-induced target discontinuity produces a pronounced acceleration spike at handover, as shown in Figure 3.

4 RRTC: Real-Time Execution of Relative-Action Policies

We present **RRTC**, a framework for real-time execution of relative-action policies $\pi(\mathcal{A}_t \mid o_t)$, focusing on flow-based policies as the policy backbone. As illustrated in Figure 4, RRTC explicitly accounts for tracking-error-induced changes in the action anchor to maintain consistent absolute targets across chunk transitions. It consists of three components: tracking-error-aware chunk re-anchoring, tracking-error-conditioned relative-action policy, and policy training with pseudo tracking errors.

4.1 Tracking-Error-Aware Chunk Re-Anchoring

To maintain cross-chunk consistency, RRTC re-expresses the target poses of the preceding chunk relative to the current reached anchor pose C_t. Following the analysis in Section 3.2 and shown in Figure 2 (c), for an overlapping timestep $\tau \in \mathcal{I}_p$, the re-anchored relative action is defined as

$$A_{t,\tau} \triangleq C_t^{-1} X_{t-h,\tau} = E_{t-h,t}\, R_{t-h,t\to\tau}, \qquad \tau \in \mathcal{I}_p, \tag{1}$$

where $E_{t-h,t} = C_t^{-1} X_{t-h,t}$ is the tracking error at the current anchor, and $R_{t-h,t\to\tau} = X_{t-h,t}^{-1} X_{t-h,\tau}$ denotes the remaining relative action of the preceding chunk.

By construction, $C_t A_{t,\tau} = X_{t-h,\tau}$, so re-anchoring changes only the relative representation of the preceding chunk while preserving its planned absolute target poses over the prefix window. RRTC therefore compensates for the tracking-error-induced anchor shift without changing these targets, as illustrated in Figure 4(a). We collect the re-anchored actions as $\mathcal{P}_t = \{A_{t,\tau}\}_{\tau\in\mathcal{I}_p}$. In the following subsection, $\mathcal{P}_t$ is kept as a fixed prefix during the generation of the new relative-action chunk, while the tracking error $E_{t-h,t}$ is separately provided to the policy as an explicit execution condition.

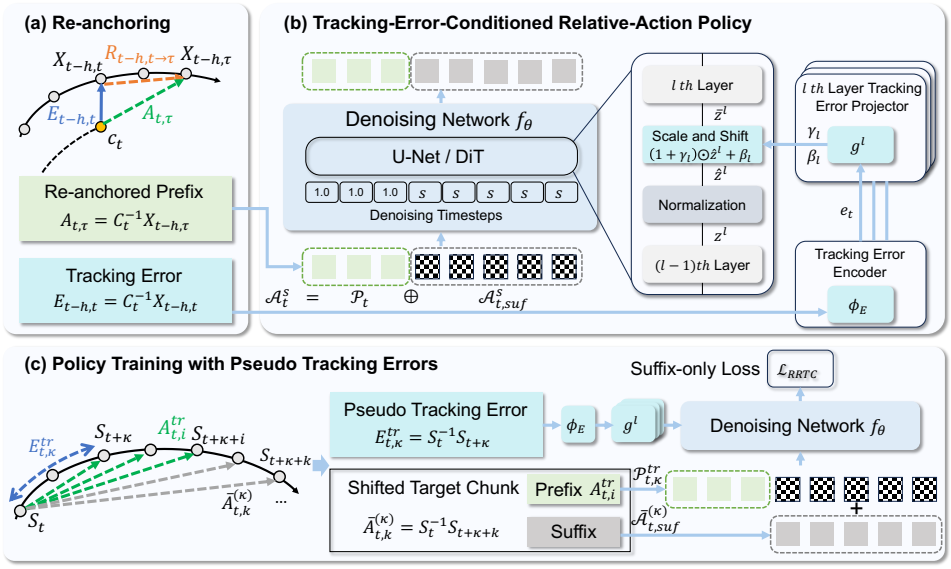


Figure 4: **Overview of RRTC.** (a) Re-anchoring preserves absolute targets under anchor shifts. (b) The policy generates an action suffix conditioned on a fixed prefix and injected tracking error. (c) Demonstration offsets provide pseudo tracking errors for training with robot-free demonstrations.

4.2 Tracking-Error-Conditioned Relative-Action Policy

RRTC conditions the relative-action policy on the latest observation o_t, the re-anchored prefix P_t, and the tracking-error condition e_t, yielding $\mathcal{A}_t \sim \pi_\theta(\cdot \mid o_t, \mathcal{P}_t, e_t)$, as shown in Figure 4 (b). For flow-based policies, π_θ is instantiated by a conditional generation model $f_\theta(\mathcal{A}_t^s, s \mid o_t, \mathcal{P}_t, e_t)$, where $\mathcal{A}_t^s$ and s denote the intermediate action state and flow timestep, respectively.

Re-Anchored Prefix Conditioning. Following TT-RTC (Black et al., 2025b), RRTC adopts hard-prefix conditioning for action generation. Unlike directly using the actions within the prefix window of the preceding chunk, RRTC first re-anchors these actions according to the tracking error, yielding the re-anchored prefix $\mathcal{P}_t = \{A_{t,\tau}\}_{\tau \in \mathcal{I}_p}$. During training, we uniformly sample the prefix horizon $L_p \sim \mathcal{U}\{0, \ldots, L_p^{\max}\}$, exposing the policy to different prefix lengths that may arise from varying inference delays. Let $\mathcal{A}_t^s$ denote the intermediate state of the new action chunk at flow step s. RRTC fixes its first L_p actions to the re-anchored prefix P_t, while only the remaining $H - L_p$ suffix actions are noised and generated. Specifically, as shown in Figure 4 (b),

$$\mathcal{A}_t^s = \underbrace{\mathcal{P}_t}_{\text{fixed prefix}} \oplus \underbrace{\mathcal{A}_{t,\text{suf}}^s}_{\text{generated suffix}}, \tag{2}$$

where $\oplus$ denotes temporal concatenation and $\mathcal{A}_{t,\text{suf}}^s$ denotes the intermediate state of the suffix at generation step s. Thus, $\mathcal{P}_t$ remains unchanged throughout the generation process, while $\mathcal{A}_{t,\text{suf}}^s$ is progressively denoised or transported to produce the suffix $\mathcal{A}_{t,\text{suf}}$. The final output action chunk is therefore $\mathcal{A}_t = \mathcal{P}_t \oplus \mathcal{A}_{t,\text{suf}}$. Since the fixed prefix satisfies $C_t A_{t,\tau} = X_{t-h,\tau}$ for $\tau \in \mathcal{I}_p$, preserving $\mathcal{P}_t$ throughout generation preserves the absolute targets planned by the preceding chunk over the prefix window. Thus, it ensures that the new chunk begins with a absolute trajectory that is consistent with the preceding chunk after absolute target recovery.

Tracking-Error Injection. Re-anchored prefix conditioning alone does not explicitly condition suffix generation on the current tracking error. Although the tracking error is already contained in the re-anchored prefix, it enters the generation model only through the fixed action tokens. The model must therefore extract this information from the prefix representation for generating the remaining actions. To provide a direct conditioning pathway, RRTC introduces a tracking-error injection module that explicitly represents the tracking error as a global condition. Specifically, we encode the tracking-error transform as $e_t = \phi_E(E_{t-h,t})$, where $\phi_E(\cdot)$ maps the tracking-error to a condition embedding. We therefore treat e_t as a global execution condition and inject it into the intermediate features of the flow backbone so that the tracking error can directly modulate suffix generation throughout the network. Specifically, for the ℓ-th network block with hidden feature z^ℓ,

we obtain layer-wise scale and shift modulation parameters and apply

$$\left(\gamma_e^\ell, \beta_e^\ell\right) = g_e^\ell(e_t), \quad \tilde{z}^\ell = \left(1 + \gamma_e^\ell\right) \odot \mathrm{Norm}(z^\ell) + \beta_e^\ell, \tag{3}$$

Here, $g^\ell(\cdot)$ denotes the condition projection for the ℓ-th network block. This feature-level modulation makes the tracking error directly available throughout the generation process, allowing the flow dynamics to adapt the generated suffix to the current tracking error. This conditioning branch can be combined with the original timestep conditioning of flow-based policies and can be added to pretrained models using zero-initialized modulation parameters, thereby preserving the pretrained behavior at the beginning of adaptation. Further implementation details are provided in Appendix A.

4.3 Policy Training with Pseudo Tracking Errors

At deployment, the tracking error $E_{t-h,t}$ can be computed directly from the target pose and the reached end-effector pose, whereas robot-free demonstrations provide only reached end-effector trajectories. This creates a mismatch between the tracking-error conditions available during training and those encountered at deployment. To bridge this gap, RRTC synthesizes pseudo tracking errors as a temporal offset along a demonstration trajectory, as shown in Figure 4 (c). Consider a demonstration $\mathcal{T} = \{(o_t, S_t)\}_{t=0}^{T}$, where $S_t \in SE(3)$ denotes the demonstration trajectory. We sample a temporal offset $\kappa \sim \mathcal{U}\{0, 1, ..., \kappa_{max}\}$, retain o_t and S_t as the current observation and reached pose, and regard $S_{t+\kappa}$ as the planned handover pose. The resulting pseudo tracking-error is

$$E_{t,\kappa}^{\mathrm{tr}} = S_t^{-1} S_{t+\kappa}. \tag{4}$$

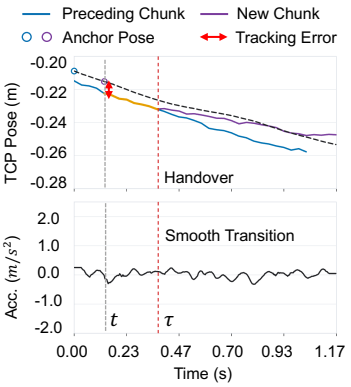


Figure 5: Execution with RRTC.

Following the same re-anchoring operation used at deployment, we construct the corresponding training prefix as $a_{t,i}^{\mathrm{tr}} = E_{t,\kappa}^{\mathrm{tr}}\left(S_{t+\kappa}^{-1} S_i\right) = S_t^{-1} S_i, i \in \{t+\kappa, ..., t+\kappa+L_p-1\}$, and denote the fixed prefix by $\mathcal{P}_{t,\kappa}^{\mathrm{tr}} = \{A_{t,i}^{\mathrm{tr}}\}_{i=t+\kappa}^{t+\kappa+L_p-1}$. The target relative-action chunk is defined as $\bar{A}_{t,j}^{(\kappa)} = S_t^{-1} S_j, j \in \{t+\kappa, \ldots, t+\kappa+H-1\}$, with $\bar{\mathcal{A}}_t^{(\kappa)} = \{\bar{A}_{t,j}^{(\kappa)}\}_{j=t+\kappa}^{t+\kappa+H-1}$. By construction, the target actions over the conditioned prefix satisfy $A_{t,i}^{\mathrm{tr}} = \bar{A}_{t,i}^{(\kappa)}$. We therefore partition the training target into the fixed prefix and the remaining suffix: $\bar{\mathcal{A}}_t^{(\kappa)} = \mathcal{P}_{t,\kappa}^{\mathrm{tr}} \oplus \bar{\mathcal{A}}_{t,\mathrm{suf}}^{(\kappa)}$, where $\oplus$ denotes temporal concatenation and $\bar{\mathcal{A}}_{t,\mathrm{suf}}^{(\kappa)}$ denotes the action suffix to be generated. Following the hard-prefix conditioning scheme described in Section 4.2, $\mathcal{P}_{t,\kappa}^{\mathrm{tr}}$ remains clean and fixed throughout generation, while only the suffix is noised. RRTC retains the original diffusion or flow-matching objective of the underlying policy, while applying the prediction loss only to the generated suffix. Let $\mathcal{M}_{\mathrm{suf}}$ denote a binary mask that is zero over the fixed prefix and one over the generated suffix. The training objective is

$$\mathcal{L}_{\mathrm{RRTC}} = \mathbb{E}_{\mathcal{T},t,\kappa,s}\left[\left\|\mathcal{M}_{\mathrm{suf}} \odot \left(f_\theta\left(\mathcal{A}_t^s, s \mid o_t, e_{t,\kappa}^{\mathrm{tr}}\right) - y_t^s\right)\right\|_2^2\right], \tag{5}$$

where y_t^s denotes the standard denoising target. The prefix serves only as a fixed conditioning context and is excluded from the loss. By sampling different temporal offsets κ, RRTC exposes the policy to a range of pseudo tracking-error conditions. We validate the pseudo tracking error design using real-robot replay data with diverse controller setups (Appendix B). Matching each measured tracking error to the closest temporal-offset deviation explains 91.8% of the tracking error, while the actual tracking error of 99.8% of replay samples lies within the range covered by the temporal offsets $\kappa \in \{0, \ldots, 4\}$.

Combining all the above designs, our method compensates for the tracking-error-induced anchor shift and restores cross-chunk target consistency, as shown in Figure 5. Despite the tracking error between the planned and reached anchor poses, the absolute target poses of the preceding and new chunks remain continuous at handover, avoiding acceleration jitter and enabling faster and more reliable robot execution.

5 Experiments

Evaluation Metrics. We evaluate RRTC in terms of task success, execution efficiency, and motion smoothness. We report *Success Rate* (**SR** ↑) for task reliability. For execution efficiency, we

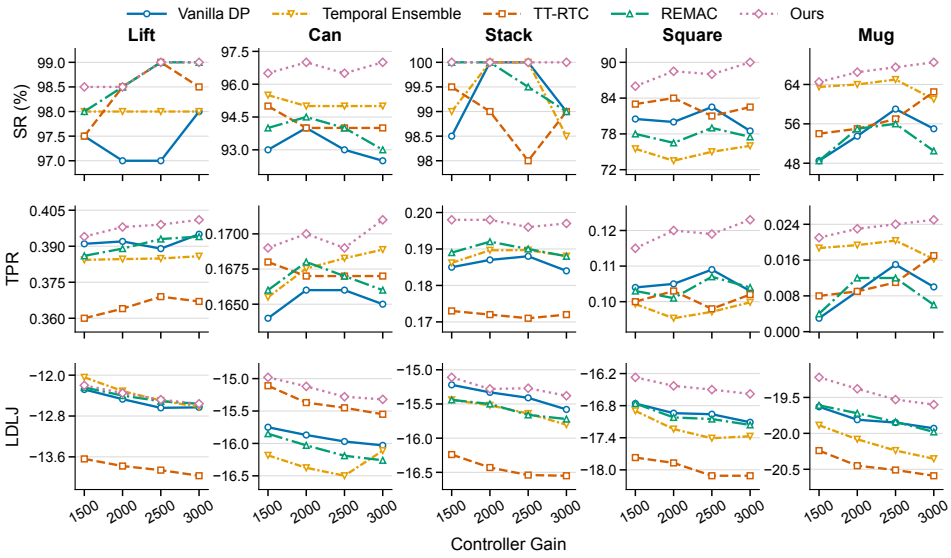


Figure 6: **Simulation performance under different controller gains.** Columns correspond to the five manipulation tasks, and rows report success rate (**SR**), throughput-with-regret (**TPR**), and log dimensionless jerk (**LDLJ**), respectively. Higher values indicate better performance for all metrics.

use *Throughput-with-Regret* (**TPR** ↑) (Arachchige et al., 2025) in simulation, while reporting the average completion time (**Time** ↓) over successful rollouts in real-world experiments. For motion smoothness, we report *Log Dimensionless Jerk* (**LDLJ** ↑) (Balasubramanian et al., 2011). Detailed definitions are provided in Appendix C.1.

Baselines. (1) **Naive Async**, which directly performs asynchronous chunk execution without explicit cross-chunk consistency handling; (2) **Temporal Ensemble** (Zhao et al., 2023), which smooths execution by averaging overlapping actions from recent chunks; (3) **TT-RTC** (Black et al., 2025b), which conditions action generation on the prefix inherited from the preceding chunk; and (4) **REMAC** (Wang et al., 2026b), which addresses inter- and intra-chunk inconsistency through masked action chunking. None of these methods explicitly models tracking-error-induced anchor shifts under relative-action policies. Detailed descriptions are provided in Appendix C.2.

5.1 Simulation Environment

Experiment Setup. We evaluate RRTC on the Robomimic (Mandlekar et al., 2021) and MimicGen (Mandlekar et al., 2023) benchmarks across five tasks: *Lift*, *Can*, *Stack*, *Square*, and *Mug*. These tasks span basic pick-and-place, precision-sensitive placement, and longer-horizon manipulation; detailed task descriptions are provided in Appendix D.1. We use Diffusion Policy (Chi et al., 2025) as the policy backbone for all methods and convert the demonstrated end-effector trajectories into relative-action representations. Each policy is trained with two random seeds and evaluated over 100 rollouts per seed. For all controller, execution-speed, and inference-delay sweeps, we use the same trained policy and vary only the corresponding execution setting. Detailed execution parameters and implementation settings are provided in Appendix D.2.

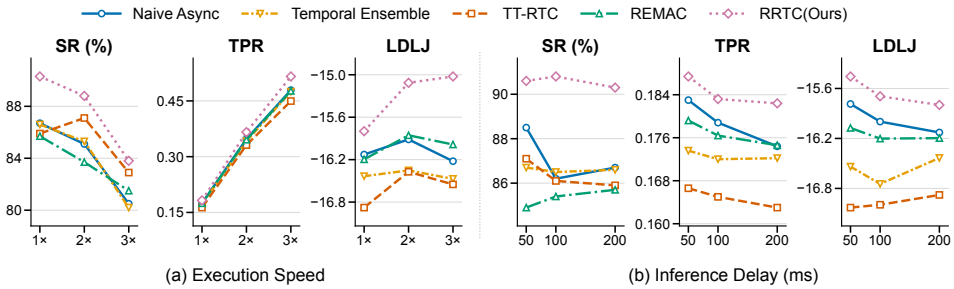


Figure 7: **Average performance across the five tasks** under (a) different execution speeds and (b) different inference delays. All other execution and controller settings are fixed within each sweep.

Results. We evaluate RRTC under diverse tracking-error distributions induced by two complementary factors: controller gains and execution speeds. Varying the controller gain alters the resulting tracking behavior (Appendix D.3.1), while increasing the execution speed can also shift the tracking-error distribution. We further vary the inference delay to evaluate robustness to computation latency.

RRTC performs consistently across controller gains. We vary the controller gain over 1500, 2000, 2500, and 3000 to induce different tracking-error distributions. As shown in Figure 6, RRTC consistently maintains strong SR, TPR, and LDLJ. Averaged across tasks and gains, RRTC reaches an SR of 90.1%, compared with 86.3% for TT-RTC and 86.2% for Temporal Ensemble. The gains are modest on Lift and Stack, where success rates are already near saturation, but become larger on the more challenging Square and Mug tasks. Relative to the strongest baseline, the gain-averaged SR improvements are 5.5 and 3.4 percentage points, respectively. By mitigating cross-chunk inconsistency at handover, RRTC reduces motion jitter and unnecessary corrective movements, enabling more continuous trajectory execution and improving both task robustness and execution efficiency.

RRTC sustains its advantage under speed-induced tracking-error shifts. We vary the execution speed while keeping the controller fixed (Appendix D.3.2 for details), thereby inducing different tracking-error distributions under the same controller. Across the tested speeds, RRTC maintains strong overall results. As shown in Figure 7(a), RRTC achieves the highest average SR, reaching 90.3%, 88.8%, and 83.8% at $1\times$, $2\times$, and $3\times$, respectively. At $3\times$, its task-averaged TPR is 0.516, a 7.7% relative improvement over the strongest TPR baseline, while its task-averaged LDLJ is 0.96 higher than the strongest LDLJ baseline. Complete per-task results are provided in Appendix D.4.1.

RRTC remains effective under varying inference delays. We vary the inference delay over 50, 100, and 200 ms while keeping the controller and execution speed fixed, to evaluate robustness to latency during real-time execution. As shown in Figure 7(b), RRTC achieves the highest task-averaged SR, TPR, and LDLJ at each tested delay. Complete per-task results are provided in Appendix D.4.2.

Ablation Study. We ablate the major components of RRTC in Table 1. Removing re-anchoring causes substantial performance degradation, reducing SR from 90.3% to 85.3% and LDLJ from -15.80 to -16.88, highlighting the importance of compensating for tracking-error-induced anchor shifts. Removing prefix conditioning, tracking-error injection, or pseudo-error training reduces SR to 87.0%, 87.5%, and 86.4%, respectively, showing that these components provide complementary benefits. Moreover, adding re-anchoring alone to TT-RTC or REMAC achieves only 85.6% and 84.2% SR, respectively, remaining below the full RRTC and indicating that re-anchoring is necessary but insufficient by itself. Detailed descriptions and per-task results are provided in Appendix D.4.3.

Table 1: Average ablation results.

Variant	SR↑	TPR↑	LDLJ↑
Full RRTC	**90.3**	**0.182**	**-15.80**
w/o Re-anchoring	85.3	0.171	-16.88
w/o Prefix Cond.	87.0	0.170	-16.22
w/o TE Injection	87.5	0.172	-16.00
w/o Pseudo-error	86.4	0.168	-16.16
TT-RTC + Re-anchoring	85.6	0.165	-16.98
REMAC + Re-anchoring	84.2	0.171	-16.31

5.2 Real-World Evaluation

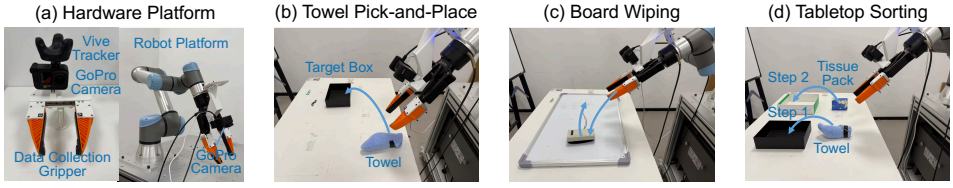


Figure 8: **Real-world setup.** (a) UMI-style robot-free data collection and the physical robot platform. (b)–(d) Evaluation tasks: Towel Pick-and-Place, Board Wiping, and Tabletop Sorting.

Experiment Setup. We collect demonstrations using a UMI gripper, with an HTC Vive tracker (Borges et al., 2018) for gripper-pose tracking. For physical execution, we use a UR7e arm equipped with a Robotiq 2F-140 gripper, as shown in Figure 8(a). We use $\pi_{0.5}$ (Physical Intelligence et al., 2025) as the base policy and fine-tune only the action expert. We evaluate three complementary tasks, as shown in Figure 8(b)–(d): *(1) Towel Pick-and-Place*, which requires grasping and transporting a deformable towel to a target region; *(2) Board Wiping*, which requires maintaining contact while wiping a marked line on a whiteboard; and *(3) Tabletop Sorting*, which requires

sequentially picking and placing two objects into their corresponding target boxes. These tasks respectively cover deformable, contact-rich, and multi-step manipulation. For each task, we collect 100 demonstration episodes for policy adaptation and perform 30 evaluation rollouts. Detailed task definitions and success criteria are provided in Appendix E.1.

Table 2: Real-robot results. Time and LDLJ are reported as mean ± sample standard deviation over successful trials. The best result in each row is shown in bold.

Task	Metric	Naive Async	Temporal Ensemble	TT-RTC	REMAC	RRTC (Ours)
Pick-and-place	Success (%) ↑	**100.0**	**100.0**	**100.0**	96.7	**100.0**
	Time (s) ↓	12.46 ± 1.28	10.54 ± 0.72	11.09 ± 0.58	12.84 ± 1.58	$\mathbf{8.23 \pm 0.60}$
	LDLJ ↑	-27.49 ± 0.72	-26.63 ± 0.68	-26.95 ± 0.45	-27.46 ± 0.55	$\mathbf{-26.26 \pm 0.66}$
Board Wiping	Success (%) ↑	50.0	46.7	66.7	43.3	**86.7**
	Time (s) ↓	12.80 ± 2.45	16.75 ± 1.91	11.90 ± 2.93	14.41 ± 1.85	$\mathbf{7.75 \pm 1.23}$
	LDLJ ↑	-27.40 ± 0.71	-28.25 ± 0.72	-27.00 ± 0.94	-27.54 ± 0.58	$\mathbf{-25.66 \pm 0.77}$
Tabletop Sorting	Success (%) ↑	66.7	50.0	70.0	73.3	**80.0**
	Time (s) ↓	39.89 ± 3.30	42.72 ± 9.11	39.42 ± 4.42	42.39 ± 2.77	$\mathbf{25.56 \pm 2.48}$
	LDLJ ↑	-31.41 ± 0.57	-31.45 ± 1.24	-31.37 ± 0.62	-31.46 ± 0.55	$\mathbf{-30.07 \pm 0.48}$

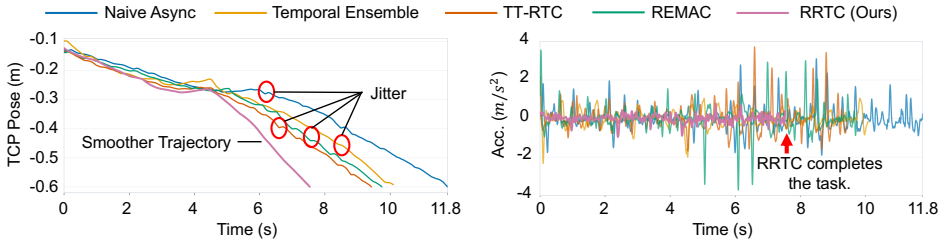


Figure 9: **Real-robot rollout trajectories.** Pose along the x-axis (left) and the corresponding acceleration (right) during a representative Towel Pick-and-Place rollout. Each trajectory is shown until the corresponding rollout completes; the horizontal axis spans the longest rollout.

Results. Table 2 summarizes the results. RRTC achieves strong performance across task success, execution efficiency, and motion smoothness. On *Towel Pick-and-Place*, where success rates are already saturated for most methods, RRTC reduces the completion time to $8.23\,\mathrm{s}$, compared with 10.54–$12.84\,\mathrm{s}$ for the baselines, while achieving the best LDLJ. The advantage is more pronounced on the more challenging tasks. On *Board Wiping*, RRTC achieves an 86.7% success rate, outperforming the strongest baseline by 20.0 percentage points, while reducing the completion time to $7.75\,\mathrm{s}$. On *Tabletop Sorting*, RRTC likewise achieves the highest success rate of 80.0%, together with the shortest completion time of $25.56\,\mathrm{s}$ and the best LDLJ. By mitigating cross-chunk discontinuities, RRTC reduces abrupt corrective motions that interrupt task progress, enabling more continuous execution and thereby shortening task completion time.

Figure 9 further illustrates this behavior through a representative Towel Pick-and-Place rollout. Compared with the baselines, RRTC produces a smoother TCP trajectory with fewer abrupt changes and substantially smaller acceleration fluctuations. Its peak absolute acceleration is approximately $0.96\,\mathrm{m/s^2}$, compared with 2.32–$3.72\,\mathrm{m/s^2}$ for the baselines. This trajectory-level comparison further illustrates that RRTC mitigates abrupt motion changes during real-time chunk transitions.

6 Conclusion and Future Work

We study real-time execution of relative-action robot policies and show that aligning overlapping relative actions alone does not guarantee cross-chunk consistency under controller tracking error. To address this problem, we propose RRTC, which combines tracking-error-aware re-anchoring, conditioned action generation, and pseudo tracking-error training. Experiments demonstrate smoother and more robust execution across diverse setups. More broadly, our results highlight the importance of accounting for action representation when designing real-time chunking methods.

RRTC focuses on generating smooth and consistent target poses across action chunks, but does not explicitly reason about contact forces during execution. Similar to RTC, TT-RTC, and REMAC, it assumes that the robot controller can safely track the generated target poses. However, the robot may be physically prevented from reaching a pose. Combining RRTC with compliant or impedance control, force-aware safety mechanisms, or contact-aware policies is therefore an important direction for future work.

AI USE STATEMENT

In this work, we used generative AI tools to assist in the writing of proofs, design or provide feedback on research methodology or experiments, implement methods, assist with translation, support qualitative and thematic data analysis, and interpret results. We did not use generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, or clean and reformat dataset. Additionally, we used generative AI tools for brainstorming, literature-search assistance, drafting and editing parts of the manuscript, and improving its clarity and readability. We have reviewed all AI-assisted work. All AI-assisted suggestions and generated text were critically reviewed, revised, and verified by the authors, including checking technical claims and relevant literature against the original sources. LLM-generated code was verified and tested for correctness by authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work strictly adheres to the ICLR Code of Ethics. This work focuses on robot learning algorithms and system design, using robot demonstration data collected by human operators. All experiments are conducted in controlled laboratory environments and are intended solely for research purposes. The study does not involve human-subject research, and we do not anticipate concerns related to harmful applications, discrimination, bias, fairness, privacy, security, legal compliance, or research integrity. Any data released with this work will be carefully reviewed and shared with appropriate ethical and privacy considerations. We also disclose and appropriately manage any potential conflicts of interest or sponsorship.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the proposed RRTC framework, training procedure, and implementation settings in the main paper and appendix. The experimental setups, benchmark tasks, evaluation protocols, controller configurations, and metrics are also documented in detail to facilitate reproduction of our simulation and real-robot experiments. Additional per-task results and ablation details are included in the appendix. An anonymous implementation and the relevant experimental configurations are provided in the paper.

REFERENCES

Nadun Ranawaka Arachchige, Zhenyang Chen, Wonsuhk Jung, Woo Chul Shin, Rohan Bansal, Pierre Barroso, Yu Hang He, Yingyang Celine Lin, Benjamin Joffe, Shreyas Kousik, et al. Sail: Faster-than-demonstration execution of imitation learning policies. *arXiv preprint arXiv:2506.11948*, 2025.

Sivakumar Balasubramanian, Alejandro Melendez-Calderon, and Etienne Burdet. A robust and sensitive metric for quantifying movement smoothness. *IEEE transactions on biomedical engineering*, 59(8):2126–2136, 2011.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*, 2025a.

Kevin Black, Allen Z Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025b.

Miguel Borges, Andrew Symington, Brian Coltin, Trey Smith, and Rodrigo Ventura. Htc vive: Analysis and accuracy improvement. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2610–2615. IEEE, 2018.

Yuanpei Chen, Chen Wang, Li Fei-Fei, and C Karen Liu. Sequential dexterity: Chaining dexterous policies for long-horizon manipulation. *arXiv preprint arXiv:2309.00987*, 2023.

Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. *arXiv preprint arXiv:2402.10329*, 2024.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

DM Dawson, Z Qu, FL Lewis, and JF Dorsey. Robust control for the tracking of robot motion. *International Journal of Control*, 52(3):581–595, 1990.

Yuchun Feng, Jinliang Zheng, Zhihao Wang, Dongxiu Liu, Jianxiong Li, Jiangmiao Pang, Tai Wang, and Xianyuan Zhan. Demystifying action space design for robotic manipulation policies. *arXiv preprint arXiv:2602.23408*, 2026.

Zipeng Fu, Tony Z Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Gregor Klančar and Igor Škrjanc. Tracking-error model-based predictive control for mobile robots in real time. *Robotics and autonomous systems*, 55(6):460–469, 2007.

Xingyu Lin, Carl Qi, Yunchu Zhang, Zhiao Huang, Katerina Fragkiadaki, Yunzhu Li, Chuang Gan, and David Held. Planning with spatial-temporal abstraction from point clouds for deformable object manipulation. *arXiv preprint arXiv:2210.15751*, 2022.

Kehui Liu, Zhongjie Jia, Yang Li, Pengan Chen, Song Liu, Xin Liu, Pingrui Zhang, Haoming Song, Xinyi Ye, Nieqing Cao, et al. Fastumi-100k: Advancing data-driven robotic manipulation with a large-scale umi-style dataset. *arXiv preprint arXiv:2510.08022*, 2025.

Songming Liu, Bangguo Li, Kai Ma, Lingxuan Wu, Hengkai Tan, Xiao Ouyang, Hang Su, and Jun Zhu. Rdt2: Exploring the scaling limit of umi data towards zero-shot cross-embodiment generalization. *arXiv preprint arXiv:2602.03310*, 2026.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. *arXiv preprint arXiv:2310.17596*, 2023.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. pi0.5: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Alireza Ranjbar, Ngo Anh Vien, Hanna Ziesche, Joschka Boedecker, and Gerhard Neumann. Residual feedback learning for contact-rich manipulation tasks with uncertainty. In *2021 IEEE/RSJ International conference on intelligent robots and systems (IROS)*, pp. 2383–2390. IEEE, 2021.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.

A Universal Robots and S Energivej. Technical specification, 2025.

Dian Wang, Boce Hu, Shuran Song, Robin Walters, and Robert Platt. A practical guide for incorporating symmetry in diffusion policy. *Advances in Neural Information Processing Systems*, 38: 94105–94130, 2026a.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Yuzhang Shang, Ramana Kompella, and Gaowen Liu. Real-time robot execution with masked action chunking. In *International Conference on Learning Representations*, volume 2026, pp. 120161–120179, 2026b.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3813–3824. IEEE, 2023.

Mengfei Zhao, Chenxi Jiang, Tuo An, Jindou Jia, and Jianfei Yang. Apex: Adaptive policy execution for precise manipulation. *arXiv preprint arXiv:2606.16504*, 2026.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Abhiram Maddukuri, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

A Details of the Tracking-Error Injection Module

Tracking-Error Injection with Denoising-Step Conditioning. Section 4.2 describes the tracking-error injection mechanism in isolation to highlight how tracking error modulates intermediate network features. For backbones that already provide a generation-step conditioning pathway (Ho et al., 2020; Perez et al., 2018; Peebles & Xie, 2023), RRTC incorporates the tracking-error embedding into this existing pathway rather than introducing a separate modulation branch.

Following Section 4.1, we define $E_{t-h,t} = C_t^{-1} X_{t-h,t}$ and extract its Lie-algebra coordinates $(\mathbf{v}_t, \boldsymbol{\omega}_t) \in \mathbb{R}^6$. We normalize these components and append their magnitudes:

$$\mathbf{r}_t = \left[\frac{\mathbf{v}_t}{s_p}, \frac{\boldsymbol{\omega}_t}{s_R}, \frac{\|\mathbf{v}_t\|_2}{s_p}, \frac{\|\boldsymbol{\omega}_t\|_2}{s_R} \right] \in \mathbb{R}^8. \tag{6}$$

We set $s_p = 0.05\,\mathrm{m}$ and $s_R = 0.5\,\mathrm{rad}$ to match the physical action scales of the Robosuite `OSC_POSE` controller (Zhu et al., 2020) in Robomimic and Mimicgen, where a unit normalized action corresponds to a maximum per-step translation of $0.05\,\mathrm{m}$ and rotation of $0.5\,\mathrm{rad}$, respectively. Non-pose components, such as gripper commands, are excluded.

A two-layer MLP shared across blocks encodes the tracking error:

$$e_t = \phi_E(E_{t-h,t}) = W_{E,2}\,\mathrm{SiLU}\left(W_{E,1}\mathbf{r}_t + b_{E,1}\right) + b_{E,2}. \tag{7}$$

Its output matches the dimension of the original timestep condition $\phi_s(s)$ and is added to it before computing the block-specific scale and shift:

$$\left(\gamma_e^\ell, \beta_e^\ell\right) = g_e^\ell(\phi_s(s) + e_t)\,, \tag{8}$$

where g_e^ℓ is the backbone's existing modulation projection, with other conditioning inputs omitted for brevity. The error encoder has dimensions $8 \to 256 \to 256$ for Diffusion Policy and $8 \to 1024 \to 1024$ for $\pi_{0.5}$. In Diffusion Policy, the combined embedding enters the U-Net (Ronneberger et al., 2015) condition encoders alongside the observation features. In $\pi_{0.5}$, it enters the Action Expert's AdaRMSNorm pathway based on RMSNorm (Zhang & Sennrich, 2019).

Policy Training with Pretrained Backbones. Similar to TT-RTC (Black et al., 2025b), RRTC supports starting from a pretrained relative-action policy and introduces re-anchored prefix conditioning and tracking-error injection during fine-tuning. To ensure that the added error encoder initially preserves the pretrained conditioning, we zero-initialize its final layer: $W_{E,2} = 0$ and $b_{E,2} = 0$. This follows the principle of introducing new conditioning branches with zero initial contribution (Zhang et al., 2023).

Consequently, $e_t = 0$ for any tracking-error input, and Eq. 8 reduces to

$$\left(\gamma_e^\ell, \beta_e^\ell\right) = g_e^\ell(\phi_s(s))\,. \tag{9}$$

The modulation parameters therefore retain their pretrained values. For otherwise identical inputs and prefix conditioning, the added tracking-error pathway leaves the model's initial output unchanged. Fine-tuning then learns how tracking errors should adjust the timestep condition to guide action generation.

B Validation of Pseudo Tracking Errors

RRTC synthesizes pseudo tracking errors by applying temporal offsets to demonstration trajectories, $E^{\mathrm{tr}}_{t,\kappa} = S_t^{-1} S_{t+\kappa}$, where $\kappa \sim \mathcal{U}\{0, 1, \ldots, \kappa_{\max}\}$. We set $\kappa_{\max} = 4$ so that the temporal-offset pseudo error range better covers the tracking error observed during robot execution.

We empirically examine whether pseudo tracking error covers the distribution of physical tracking errors using real-robot rollout data. Following the setup in Appendix E.1, we collect four Towel Pick-and-Place demonstrations with a UMI handheld gripper and replay each trajectory on a Universal Robots UR7e arm (Universal Robots & Energivej, 2025) under four controller gains, $\{100, 200, 300, 400\}$, thereby inducing diverse tracking-error distributions. For each replay, we record the commanded and reached end-effector poses, resulting in 16 replay trajectories.

Let $T_t^{\mathrm{cmd}}, T_t^{\mathrm{real}} \in SE(3)$ denote the commanded and reached end-effector poses at replay sample t. During replay, robot commands are streamed at approximately 500 Hz, corresponding to a control interval of about 2 ms. The commanded and reached poses are time-aligned before computing the tracking error,

$$E_t^{\mathrm{real}} = \left(T_t^{\mathrm{real}}\right)^{-1} T_t^{\mathrm{cmd}}. \tag{10}$$

Tracking Error Explained by Training Offsets. We first examine whether the translational tracking errors observed during physical replay can be approximated by the temporal offsets used during RRTC training. Because the demonstration trajectories are recorded at nonuniform time intervals, while robot commands during replay are streamed at approximately $f_{\mathrm{ctrl}} = 500$ Hz, the same training offset κ corresponds to different physical durations along the trajectory.

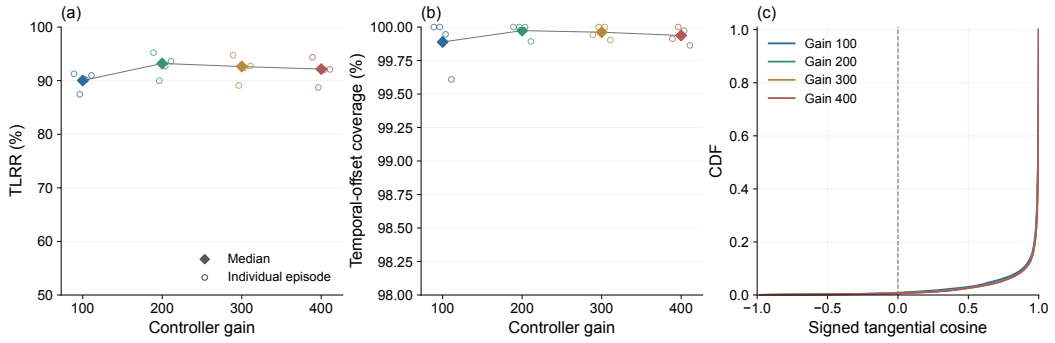


Figure 10: **Validation of pseudo tracking errors across controller gains.** (a) Median Temporal-Lag Residual Reduction (TLRR). (b) Fraction of fitted physical lags covered by the temporal-offset range used during training. (c) Empirical CDFs of the signed tangential tracking-error ratio.

Let τ_i^{demo} denote the timestamp of the i-th demonstration sample, and let $i(t)$ denote the demonstration sample corresponding to replay step t. The physical duration represented by a training offset κ is

$$\Delta_{\kappa,t} = \tau_{i(t)}^{\mathrm{demo}} - \tau_{i(t)-\kappa}^{\mathrm{demo}}, \tag{11}$$

which corresponds to

$$j_{\kappa,t} = \mathrm{round}\left(f_{\mathrm{ctrl}}\Delta_{\kappa,t}\right) \tag{12}$$

replay control steps. For $\kappa \in \{0, 1, 2, 3, 4\}$, the median physical durations are approximately 0, 66.5, 132.4, 167.1, and 233.5 ms, respectively.

We next compare the tracking error measured during real-robot replay with the error induced by these temporal offsets. For an end-effector pose $T \in SE(3)$, let $\mathrm{trans}(T) \in \mathbb{R}^3$ denote its translation component. We define

$$\mathbf{p}_t^{\mathrm{cmd}} = \mathrm{trans}\left(T_t^{\mathrm{cmd}}\right), \qquad \mathbf{p}_t^{\mathrm{real}} = \mathrm{trans}\left(T_t^{\mathrm{real}}\right), \tag{13}$$

where $\mathbf{p}_t^{\mathrm{cmd}}$ and $\mathbf{p}_t^{\mathrm{real}}$ are the commanded and reached end-effector positions at replay step t.

The measured translational tracking-error vector is

$$\mathbf{e}_t^{\mathrm{real}} = \mathbf{p}_t^{\mathrm{cmd}} - \mathbf{p}_t^{\mathrm{real}}, \tag{14}$$

with magnitude

$$d_t^{\mathrm{real}} = \left\|\mathbf{e}_t^{\mathrm{real}}\right\|_2. \tag{15}$$

For a training offset κ, we define the corresponding pseudo tracking-error vector on the replay trajectory as

$$\mathbf{e}_{t,\kappa}^{\mathrm{pseudo}} = \mathbf{p}_t^{\mathrm{cmd}} - \mathbf{p}_{t-j_{\kappa,t}}^{\mathrm{cmd}}. \tag{16}$$

This vector represents the translational deviation that would arise if the robot lagged behind the commanded trajectory by the physical duration associated with κ.

For each replay step, we find the training offset whose pseudo tracking error is closest to the measured tracking error:

$$\kappa_t^\star = \underset{\kappa\in\{0,\ldots,\kappa_{\max}\}}{\arg\min} \left\|\mathbf{e}_t^{\mathrm{real}} - \mathbf{e}_{t,\kappa}^{\mathrm{pseudo}}\right\|_2. \tag{17}$$

The remaining translational error after accounting for this temporal offset is

$$d_t^{\mathrm{res}} = \left\|\mathbf{e}_t^{\mathrm{real}} - \mathbf{e}_{t,\kappa_t^\star}^{\mathrm{pseudo}}\right\|_2. \tag{18}$$

We quantify how much the temporal-offset approximation reduces the measured tracking error using the *Temporal-Lag Residual Reduction* (TLRR):

$$\mathrm{TLRR}_t = 1 - \frac{d_t^{\mathrm{res}}}{\max(d_t^{\mathrm{real}}, \epsilon)}, \tag{19}$$

where ϵ is a small constant used to avoid numerical instability for near-zero tracking errors. A TLRR of 1 indicates that the measured translational tracking error is fully explained by one of the temporal offsets, while a value of 0 indicates no reduction in residual error.

Across all replay trajectories, the median measured translational error is 7.67 mm and the median residual after temporal-offset matching is 0.45 mm. The median per-sample TLRR is 91.8%. The corresponding median TLRR values are 89.9%, 93.3%, 92.6%, and 92.1% for controller gains 100, 200, 300, and 400, respectively, as shown in Figure 10(a). These results show that short temporal offsets explain a large fraction of the translational tracking error across the evaluated controller settings, supporting their use as a robot-free approximation of lag-dominated execution errors.

Coverage of the Training Offset Range. We next examine whether the temporal-offset range used during training is sufficient to cover the physical execution lags observed during robot replay.

For each replay sample, we first estimate its physical lag without restricting the search to the offsets used during training. Specifically, we search over a wider replay window and find the command that best matches the reached end-effector position:

$$j_t^\star = \arg\min_{j \in \mathcal{J}} \left\| \mathbf{p}_t^{\mathrm{real}} - \mathbf{p}_{t-j}^{\mathrm{cmd}} \right\|_2, \tag{20}$$

where $\mathcal{J}$ denotes the wider search window. Since $j_t^\star$ is measured in approximately 500-Hz replay control steps, whereas the training offset κ is measured in demonstration steps with nonuniform timestamps, we compare them in physical time. The estimated replay lag is

$$\ell_t = \tau_t^{\mathrm{rep}} - \tau_{t-j_t^\star}^{\mathrm{rep}}, \tag{21}$$

where τ_t^{rep} denotes the replay timestamp. Thus, ℓ_t measures the temporal delay between the reached robot motion and its best-matching point on the commanded trajectory.

During training, pseudo tracking errors are synthesized using $\kappa \in \{0, \ldots, \kappa_{\max}\}$ with $\kappa_{\max} = 4$. Because the demonstration sampling intervals are nonuniform, the maximum physical duration represented by this offset depends on the local trajectory phase:

$$\Delta_{\max,t}^{\mathrm{train}} = \tau_{i(t)}^{\mathrm{demo}} - \tau_{i(t)-\kappa_{\max}}^{\mathrm{demo}}, \tag{22}$$

where $i(t)$ denotes the demonstration sample corresponding to replay step t. Across the analyzed samples, the median value of $\Delta_{\max,t}^{\mathrm{train}}$ is approximately 233.5 ms.

We define the empirical *Temporal-Offset Coverage* as

$$\widehat{C}_{\kappa_{\max}} = \frac{1}{N} \sum_{t=1}^{N} \mathbf{1}\left[\ell_t \le \Delta_{\max,t}^{\mathrm{train}}\right]. \tag{23}$$

This metric measures the fraction of estimated physical execution lags that fall within the temporal range represented by the training offsets. Across all replay samples, the coverage is 99.8%, with gain-specific values of 99.6%, 99.9%, 99.9%, and 99.8% for controller gains 100, 200, 300, and 400, respectively, as shown in Figure 10(b). These results indicate that the temporal range represented by $\kappa_{\max} = 4$ covers nearly all execution lags observed under the evaluated controller settings.

Alignment with the Motion Direction. We further examine whether the translational tracking error is aligned with the local direction of the commanded trajectory, as expected under a temporal-lag interpretation. Let

$$\mathbf{e}_t^{\mathrm{real}} = \mathbf{p}_t^{\mathrm{cmd}} - \mathbf{p}_t^{\mathrm{real}} \tag{24}$$

denote the measured translational tracking-error vector, and let

$$\mathbf{u}_t = \frac{\mathbf{v}_t^{\mathrm{cmd}}}{\|\mathbf{v}_t^{\mathrm{cmd}}\|_2} \tag{25}$$

denote the unit tangent of the commanded trajectory at replay step t. We define the *signed tangential cosine* as

$$\gamma_t = \frac{(\mathbf{e}_t^{\mathrm{real}})^\top \mathbf{u}_t}{\|\mathbf{e}_t^{\mathrm{real}}\|_2}. \tag{26}$$

Since $\mathbf{u}_t$ is a unit vector, $\gamma_t \in [-1, 1]$ measures the signed alignment between the tracking error and the local trajectory direction. A value close to $+1$ indicates that the realized pose trails the commanded pose along the direction of motion, as expected under execution lag, whereas a negative value indicates an error pointing against the motion direction.

We exclude samples with commanded speed below 5 mm/s or translational tracking error below 0.1 mm, for which the motion or error direction is poorly defined. Across all valid samples, the median signed tangential cosine is 0.994. Moreover, 99.2% of the samples satisfy $\gamma_t > 0$, whereas only 0.8% have $\gamma_t < 0$. As shown in Figure 10(c), the distributions remain concentrated near $+1$ across all controller gains. These results show that the observed translational tracking errors are predominantly aligned with the motion direction expected under execution lag, further supporting temporal offsets as a proxy for lag-dominated execution errors.

Rotational Tracking Error. We further examine whether temporal offsets also explain the rotational component of physical tracking error. Using the geodesic distance on $SO(3)$, we define the rotational residual reduction analogously to TLRR. Across all samples, temporal-offset matching reduces the median rotational error from $0.540°$ to $0.091°$, with a median rotational residual reduction of 82.2%. The corresponding values remain stable across controller gains, ranging from 79.3% to 83.8%. These results indicate that the rotational component of the observed tracking error also exhibits a strong temporal-lag structure.

C Evaluation Details

C.1 Evaluation Metrics

We provide the detailed definitions of the evaluation metrics used in our experiments.

Success Rate (SR). Given N evaluation rollouts, we compute the success rate as

$$\mathrm{SR} = \frac{1}{N} \sum_{i=1}^{N} S_i, \tag{27}$$

where $S_i \in \{0, 1\}$ indicates whether rollout i successfully completes the task.

Throughput-with-Regret (TPR). Following (Arachchige et al., 2025), we use TPR in simulation to jointly measure task success and execution efficiency:

$$\mathrm{TPR} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{S_i}{t_i} - \frac{1 - S_i}{t^{\max}} \right), \tag{28}$$

where t_i is the execution time of rollout i, and $t^{\max}$ is the maximum allowed rollout duration. For all simulation tasks, we set $t^{\max} = 20\,\mathrm{s}$. A successful rollout receives a reward inversely proportional to its completion time, whereas a failed rollout receives a fixed penalty of $-1/t^{\max}$. TPR therefore captures the trade-off between task reliability and execution efficiency, with higher values indicating better overall performance.

Completion Time (Time). For real-world experiments, we report the average completion time over successful rollouts as a direct measure of physical execution efficiency:

$$\mathrm{Time} = \frac{1}{N_{\mathrm{succ}}} \sum_{i:S_i=1} t_i, \tag{29}$$

where $N_{\mathrm{succ}} = \sum_{i=1}^{N} S_i$ denotes the number of successful rollouts. Failed rollouts are excluded from this metric and are instead reflected by the success rate. A lower completion time indicates faster task execution.

Log Dimensionless Jerk (LDLJ). We use LDLJ (Balasubramanian et al., 2011) to quantify the smoothness of the reached end-effector motion. Let $v(t)$ denote the translational speed of the end effector over a rollout of duration T, and let

$$v_{\text{peak}} = \max_{t\in[0,T]} v(t). \tag{30}$$

The dimensionless jerk (DLJ) and its logarithmic form are computed as

$$\text{DLJ} = -\frac{T^3}{v_{\text{peak}}^2} \int_0^T \left| \frac{d^2 v(t)}{dt^2} \right|^2 dt, \tag{31}$$

$$\text{LDLJ} = -\ln |\text{DLJ}| . \tag{32}$$

The normalization reduces the dependence of the jerk measure on movement duration and speed, and a higher LDLJ corresponds to smoother motion.

For discrete trajectories, we obtain $v(t)$ from consecutive reached end-effector positions, approximate its derivatives using finite differences, and numerically integrate the squared derivative over the rollout. We report the mean LDLJ across evaluation rollouts.

C.2 Baseline Details

We provide additional details of the baselines used in our experiments. All baselines use the same policy backbone, training data, and execution settings as RRTC for a fair comparison.

Naive Asynchronous Execution (Naive Async). Naive Async directly deploys the underlying policy in an asynchronous manner. While the current action chunk is being executed, the next chunk is generated from the latest observation and is switched in once the current chunk reaches its execution horizon. No explicit mechanism is introduced to align consecutive chunks or enforce consistency at handover. We instantiate this baseline with Diffusion Policy in simulation and $\pi_{0.5}$ in the real-robot experiments. Because each newly generated relative-action chunk is anchored at the robot's current reached pose, controller tracking errors can shift the absolute targets of the new chunk relative to those planned by the preceding chunk. Naive Async does not explicitly correct this mismatch, and therefore directly exposes the execution to tracking-error-induced cross-chunk discontinuities.

Temporal Ensemble. Temporal Ensemble (Zhao et al., 2023) combines overlapping predictions from consecutive chunks using temporally varying weights, such that the contribution gradually shifts from the previous chunk to the new chunk over the overlapping region. In our implementation, the overlapping predictions are first transformed into the same absolute-action space and then averaged to obtain the executed command. While this interpolation can reduce abrupt changes at chunk boundaries, it does not guarantee that the interpolated actions remain on a valid trajectory when consecutive chunks represent different motion modes. As pointed out by RTC (Black et al., 2025a), averaging between two individually valid but divergent trajectories can produce poor intermediate actions.

Training-Time Real-Time Chunking (TT-RTC). TT-RTC (Black et al., 2025b) improves real-time chunk execution by conditioning the generation of a new action chunk on a prefix inherited from the preceding chunk. We adapt TT-RTC to our relative-action setting as follows.

During training, we directly use the relative-action demonstration trajectories without introducing pseudo tracking errors. For each training chunk, an initial segment of the demonstrated actions is used as the clean prefix and kept fixed during generation, while the remaining actions are learned as the continuation.

At inference time, when a new chunk is generated at time t, we construct its prefix from the overlapping portion of the preceding chunk. Specifically, we take the pose planned by the preceding chunk at time t as the reference and express its subsequent overlapping trajectory relative to this planned pose. Using the notation in Section 3.2, the TT-RTC prefix is therefore constructed from

$$R_{t-h,t\to\tau} = X_{t-h,t}^{-1} X_{t-h,\tau}, \tag{33}$$

for the overlapping steps τ. This prefix is then preserved during the generation of the new chunk.

The key limitation is that the reference pose $X_{t-h,t}$ is the pose *planned* by the preceding chunk, whereas the new relative-action chunk is actually anchored at the robot's reached pose C_t. When tracking error is present, $C_t \neq X_{t-h,t}$, and directly using $R_{t-h,t\to\tau}$ as the new relative-action prefix no longer preserves the same absolute targets. TT-RTC therefore handles prefix consistency, but does not explicitly correct the tracking-error-induced shift of the relative-action anchor.

In the corresponding ablation, we additionally equip TT-RTC with our re-anchoring operation while keeping the remaining TT-RTC training procedure unchanged. At inference time, instead of expressing the overlapping trajectory relative to the previously planned pose $X_{t-h,t}$, we use the reached robot pose C_t as the new anchor:

$$A_{t,\tau} = C_t^{-1} X_{t-h,\tau}. \tag{34}$$

The resulting prefix therefore preserves the same absolute targets planned by the preceding chunk, while expressing them relative to the robot's reached pose at the current chunk boundary. Since this variant already incorporates re-anchoring, its comparison with the full RRTC isolates the additional benefits of the remaining RRTC components, including tracking-error injection and pseudo-error training.

REMAC. REMAC (Wang et al., 2026b) addresses both inter-chunk discontinuity and intra-chunk inconsistency during asynchronous execution. It fine-tunes a pretrained policy using masked action chunking, where partially observed action chunks are used to train the policy to generate corrective continuations under execution mismatch. At inference time, the available prefix is preserved while the remaining actions are generated.

We adapt REMAC to the relative-action setting using the same action-coordinate construction as TT-RTC described above. In the main experiments, the overlapping prefix is therefore expressed relative to the pose planned by the preceding chunk at the current time step, without explicitly accounting for the discrepancy between that planned pose and the reached pose. Consequently, controller tracking errors can still induce a shift in the absolute targets represented by the preserved prefix.

For the corresponding ablation, we additionally equip REMAC with the same re-anchoring operation described above for TT-RTC, while keeping the REMAC training and generation procedures otherwise unchanged. The overlapping prefix is therefore re-expressed relative to the reached robot pose before being used for generation. This removes the tracking-error-induced anchor mismatch and allows us to evaluate the additional benefit of re-anchoring within the REMAC framework.

D Details of Simulation Evaluation

D.1 Task Descriptions

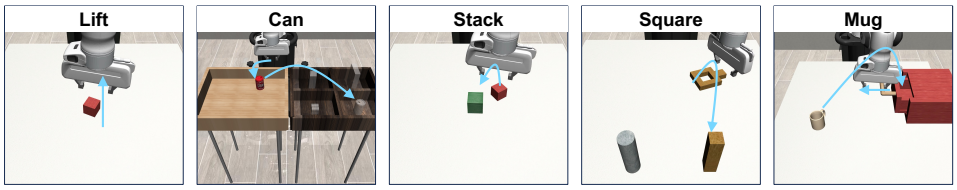


Figure 11: **Simulation tasks used in the evaluation:** Lift, Can, Stack, Square, and Mug.

We evaluate RRTC on Robomimic (Mandlekar et al., 2021) and MimicGen (Mandlekar et al., 2023) benchmarks across five manipulation tasks: Lift, Can, Stack, Square, and Mug, as shown in Figure 11. The first three tasks: Lift, Can, and Stack are relatively simple because they involve short-horizon pick-and-place behaviors with comparatively loose spatial constraints. Square and Mug are more challenging: Square requires precise position and orientation alignment during insertion, while Mug involves a longer sequence of interactions with both a movable object and an articulated drawer.

Lift. The robot is required to approach a cube on the tabletop, grasp it, and lift it vertically. At the beginning of each rollout, the cube pose is randomized within a predefined placement region. A rollout is considered successful if the cube is raised at least 4 cm above the tabletop. This short-horizon task primarily evaluates whether the policy can generate a reliable grasp and maintain stable control during lifting.

Can. The robot is required to grasp a can from the tabletop, transport it to its designated bin, and release it. The initial can pose is randomized across rollouts. A rollout is considered successful if the can is placed inside the correct bin and the gripper releases and moves away from it. Compared with Lift, this task additionally requires coordinated transport and placement, but its target region remains relatively large and tolerant to small execution errors.

Stack. The robot is required to grasp one cube, lift it, align it above a second cube, place it on top, and release it. The initial poses of both cubes are randomized. A rollout is considered successful if the upper cube is supported by and remains in contact with the lower cube after being released by the gripper. This task contains more manipulation stages than Lift and Can, but the placement target is still relatively broad compared with precision-insertion tasks.

Square. The robot is required to grasp a square nut, transport it to the corresponding square peg, align the nut opening with the peg, and insert it. The initial pose of the nut is randomized across rollouts. A rollout is considered successful if the nut is properly placed around the target peg and the gripper has released and moved away from it. This task is substantially more difficult because successful insertion requires accurate control of both end-effector position and orientation. Small tracking errors, trajectory discontinuities, or rotational misalignment can cause the nut to collide with or miss the peg.

Mug. The robot is required to open a drawer, grasp a mug from the tabletop, transport it into the drawer, release it in an upright configuration, and close the drawer. The initial mug pose is randomized across rollouts. A rollout is considered successful if the mug remains upright and in contact with the bottom of the drawer while the drawer is fully closed. This is the longest-horizon task in the benchmark: it combines multiple consecutive manipulation stages, constrained-space placement, precise object-orientation control, and interaction with an articulated object. As a result, errors can accumulate across action-chunk transitions and failures in any intermediate subtask can prevent overall success.

D.2 Simulation Execution Settings

Policy and Action Configuration. We use Diffusion Policy (Chi et al., 2025) as the policy backbone for all simulation experiments. To construct the relative-action setting, we convert the demonstrated end-effector trajectories into chunk-wise relative-action representations, where each action chunk is expressed with respect to the end-effector pose at the beginning of the chunk. Across all tasks and methods, we use a prediction horizon of $H = 32$ and an execution horizon of $h = 8$. For RRTC, the prefix horizon is set to $L_p = 4$, and the maximum temporal offset used for pseudo tracking-error training is set to $\kappa_{\max} = 4$. The demonstrations are collected at a base control frequency of 20 Hz.

The inference delay d denotes the number of old-chunk actions executed while a new chunk is generated and therefore corresponds to a physical latency of $\tau_d = d/f_c$; for example, $d = 4$ corresponds to $200\,\mathrm{ms}$ at the nominal $20\,\mathrm{Hz}$ control rate.

Training and Evaluation Protocol. Each policy is trained with two random seeds. Each trained policy is evaluated over 100 rollouts per seed, resulting in 200 rollouts per method and task. All methods use the same task setup and controller configuration under each evaluation condition. For the controller-gain, execution-speed, and inference-delay sweeps, we use the same trained policy and vary only the corresponding execution setting; no policy is retrained for individual sweep settings. Task success is determined using the criteria defined by the corresponding Robomimic or MimicGen benchmark, as detailed in Appendix D.1.

Execution-Speed Scaling. We evaluate execution speeds of $1\times$, $2\times$, and $3\times$ while keeping the trained policy and low-level controller unchanged. To realize a speedup factor s, we increase the simulation control frequency to $f_c = 20s\,\mathrm{Hz}$ while preserving the discrete action sequence, such that each action is executed for $1/(20s)$ seconds. Thus, varying the execution speed changes the tracking dynamics experienced during execution without changing the policy itself.

Inference Delay. Inference latency is a practical bottleneck for flow-based visuomotor policies. To emulate different inference delays in simulation, we delay the handover to the newly generated chunk by d control steps: while inference is assumed to be running, the controller continues executing actions from the preceding chunk for an additional d steps, and switches to the new chunk only afterward. Thus, d represents the number of control steps elapsed before a new chunk becomes available, corresponding to a physical latency of $\tau_d = d/f_c$, where f_c is the control frequency. For example, $d = 4$ corresponds to $200\,\mathrm{ms}$ at $20\,\mathrm{Hz}$. We use $d = 4$ as the default setting. Across all delay experiments, we use the same trained policy and vary only this runtime delay.

D.3 Controllers in Simulation

D.3.1 Tracking Error Distribution of Different Controllers

Table 3: Tracking error (mm) under different controller gains in simulation.

Gain	Lift			Can			Stack			Square			Mug		
	Mean	P50	P90	Mean	P50	P90	Mean	P50	P90	Mean	P50	P90	Mean	P50	P90
1500	0.81	0.54	1.62	0.91	0.70	1.88	1.32	1.11	2.47	1.03	0.72	2.02	2.17	1.01	2.64
2000	0.64	0.50	1.07	0.74	0.62	1.35	0.98	0.85	1.78	0.73	0.55	1.43	1.88	0.76	1.97
2500	0.71	0.65	1.08	0.91	0.90	1.46	0.92	0.83	1.61	0.72	0.58	1.33	1.72	0.75	1.70
3000	0.80	0.76	1.26	1.07	1.11	1.73	0.95	0.87	1.65	0.76	0.62	1.44	1.69	0.79	1.70

In simulation, we use fixed-impedance operational-space pose controllers (`OSC_POSE`) with different stiffness gains to evaluate RRTC under diverse tracking-error distributions. We fix the damping ratio at 0.5 and vary the proportional gain K_p over $\{1500, 2000, 2500, 3000\}$. In `OSC_POSE`, K_p controls the strength of the corrective operational-space force/torque applied in response to pose error, and therefore affects how closely the robot tracks the commanded end-effector trajectory. For each task and gain setting, we replay each demonstration trajectory 100 times and record the resulting tracking errors. The corresponding tracking-error distributions are summarized in Table 3.

The results show that different controller gains lead to distinct tracking-error distributions, and the effect is task-dependent rather than strictly monotonic. For example, Lift and Can achieve the lowest mean tracking errors at a gain of 2000, while Stack and Square perform best around 2500. Mug exhibits larger tracking errors overall, with mean errors ranging from 1.69 to 2.17 mm.

The differences are also reflected in the upper tail of the distributions. Across most tasks, the P90 tracking error varies noticeably with the controller gain, e.g., from 2.47 mm to 1.61 mm on Stack and from 2.02 mm to 1.33 mm on Square. These results confirm that varying the controller gain provides diverse and task-dependent tracking-error conditions for evaluating the robustness of RRTC.

D.3.2 Controller Selection for Speed and Delay Experiments

Table 4: Average success rate (%) across the five methods, including Temporal Ensemble, under different controller gains. For each task, the best controller gain is shown in **bold**.

Task	1500	2000	2500	3000
Lift	97.9	98.1	98.4	**98.5**
Can	94.8	**94.9**	94.5	94.3
Stack	99.4	**99.8**	99.5	99.1
Square	80.6	80.5	**81.1**	80.9
Mug	55.8	58.8	**60.9**	59.5

Table 5: Simulation results on five tasks under different execution speeds. We report success rate (**SR**, %), throughput-with-regret (**TPR**), and log dimensionless jerk (**LDLJ**). Higher is better for all metrics. Best and second-best results under each task and execution speed are shown in **bold** and underlined, respectively.

		Naive Async			Temporal Ensemble			TT-RTC			REMAC			RRTC(Ours)		
Task	**Speed**	**SR↑**	**TPR↑**	**LDLJ↑**	**SR↑**	**TPR↑**	**LDLJ↑**	**SR↑**	**TPR↑**	**LDLJ↑**	**SR↑**	**TPR↑**	**LDLJ↑**	**SR↑**	**TPR↑**	**LDLJ↑**
Lift	1×	98.0	0.395	-12.63	98.0	0.386	-12.62	98.5	0.367	-13.97	**99.0**	0.394	**-12.56**	**99.0**	**0.401**	**-12.56**
	2×	**100.0**	0.824	-11.60	99.5	0.792	-12.34	**100.0**	0.757	-12.88	99.5	0.818	-11.54	**100.0**	**0.843**	**-11.33**
	3×	**100.0**	1.208	-11.20	99.5	1.179	-11.80	99.0	1.101	-12.24	99.0	1.179	-11.23	97.5	**1.244**	**-10.99**
Can	1×	94.0	0.166	-15.87	95.0	0.168	-16.38	94.0	0.167	-15.37	94.5	0.168	-16.03	**97.0**	**0.170**	**-15.12**
	2×	**96.0**	0.328	-15.37	90.5	0.316	-15.48	94.5	0.321	-14.89	**96.0**	**0.331**	-15.27	95.0	0.324	**-14.53**
	3×	**97.0**	0.448	-15.18	91.5	0.455	-14.87	91.5	0.424	-15.08	93.5	0.440	-15.11	96.0	**0.476**	**-13.62**
Stack	1×	**100.0**	0.187	-15.33	**100.0**	0.190	-15.52	99.0	0.172	-16.43	**100.0**	0.192	-15.50	**100.0**	**0.198**	**-15.28**
	2×	99.5	0.359	-14.75	**100.0**	0.367	-15.15	99.0	0.336	-15.51	99.5	0.360	-14.86	**100.0**	**0.380**	**-14.02**
	3×	98.5	0.495	-14.72	97.5	0.484	-15.25	**99.5**	0.452	-15.40	98.5	0.495	-14.60	97.5	**0.516**	**-13.94**
Square	1×	82.5	0.109	-16.96	75.0	0.097	-17.41	81.0	0.098	-18.11	79.0	0.107	-17.05	**88.0**	**0.119**	**-16.50**
	2×	69.5	0.181	-17.49	67.5	0.172	-18.10	77.5	0.193	-17.84	67.5	0.180	-17.35	**78.5**	**0.219**	**-16.34**
	3×	59.0	0.205	-18.55	54.0	0.190	-19.21	65.5	0.215	-18.68	59.5	0.214	-18.18	**67.0**	**0.270**	**-16.78**
Mug	1×	59.0	0.015	-19.85	65.0	0.020	-20.24	57.0	0.011	-20.51	56.0	0.012	-19.84	**67.5**	**0.024**	**-19.53**
	2×	60.5	0.045	-20.36	69.0	0.062	-20.69	64.5	0.049	-20.74	56.0	0.039	-20.26	**70.5**	**0.064**	**-19.35**
	3×	48.0	0.039	-21.46	58.5	0.066	-21.23	59.0	0.056	-21.35	57.0	0.057	-20.81	**61.0**	**0.073**	**-19.79**

For the execution-speed and inference-delay experiments, we select a task-specific controller under the default setting of $1\times$ execution speed and 200 ms inference delay. Specifically, for each task, we compute the average SR across all evaluated methods under each controller gain and select the gain with the highest average performance. Once selected, the controller is kept fixed across all methods and all subsequent speed and delay settings for that task. The selected controller gains for each task are highlighted in bold in Table 4.

D.4 Detailed Simulation Results

D.4.1 Detailed Results under Different Execution Speeds

Table 5 reports the complete per-task results under execution speeds of $1\times$, $2\times$, and $3\times$. For each task, we keep the controller fixed to the task-specific configuration selected in Appendix D.3.2, and use the same controller across all methods and execution speeds. We also fix the inference delay to $d = 4$ control steps for all speed settings.

Overall, RRTC maintains strong performance as the execution speed increases. Averaged across the five tasks, RRTC achieves SRs of 90.3%, 88.8%, and 83.8% at $1\times$, $2\times$, and $3\times$, respectively, outperforming the baselines at all three speeds. The corresponding TPR increases from 0.182 to 0.366 and 0.516 as execution accelerates, indicating that RRTC can effectively convert faster execution into higher task throughput while maintaining competitive success rates. RRTC also achieves the highest average LDLJ at every speed, with the advantage becoming particularly clear at $2\times$ and $3\times$.

At the task level, Lift and Stack remain relatively insensitive to increased execution speed because their success rates are already close to saturation. In contrast, Square and Mug become substantially more challenging as execution accelerates. On Square, RRTC achieves SRs of 88.0%, 78.5%, and 67.0% from $1\times$ to $3\times$, consistently outperforming the other methods. Similarly, RRTC achieves the highest SR on Mug at all three speeds, including 61.0% at $3\times$. These results further show that the benefit of RRTC is most evident on tasks where faster execution induces larger tracking errors and makes cross-chunk consistency more critical.

D.4.2 Detailed Results under Different Inference Delays

Table 6 reports the complete per-task results under inference delays of 50, 100, and 200 ms. For each task, we keep the execution speed fixed at $1\times$ and use the task-specific controller selected in Appendix D.3.2. The same controller and trained policy are used across all methods and delay settings, with only the runtime inference delay varied.

Table 6: Simulation results on five tasks under different inference delays at the nominal execution speed ($1\times$). We report success rate (**SR**, %), throughput-with-regret (**TPR**), and log dimensionless jerk (**LDLJ**). Higher is better for all metrics. Best and second-best results under each task and delay are shown in **bold** and underlined, respectively.

		Naive Async			Temporal Ensemble			TT-RTC			REMAC			RRTC (Ours)		
Task	**Delay (ms)**	**SR↑**	**TPR↑**	**LDLJ↑**	**SR↑**	**TPR↑**	**LDLJ↑**	**SR↑**	**TPR↑**	**LDLJ↑**	**SR↑**	**TPR↑**	**LDLJ↑**	**SR↑**	**TPR↑**	**LDLJ↑**
Lift	50	98.0	0.420	-12.24	97.5	0.387	-12.74	99.0	0.381	-14.10	97.5	0.410	-12.39	**99.5**	**0.426**	**-12.17**
	100	97.5	0.407	-12.56	**99.0**	0.386	-12.86	98.5	0.375	-14.10	98.0	0.401	-12.62	98.5	**0.409**	**-12.54**
	200	98.0	0.395	-12.63	98.0	0.386	-12.62	98.5	0.367	-13.97	**99.0**	0.394	**-12.56**	**99.0**	**0.401**	**-12.56**
Can	50	**96.5**	0.167	-16.43	96.0	**0.169**	-16.21	89.5	0.154	-15.70	95.0	0.165	-16.57	95.0	0.168	**-15.50**
	100	94.0	0.162	-16.26	93.5	0.164	-16.29	92.5	0.163	-15.48	94.0	0.163	-16.42	**97.0**	**0.169**	**-15.18**
	200	94.0	0.166	-15.87	95.0	0.168	-16.38	94.0	0.167	-15.37	94.5	0.168	-16.03	**97.0**	**0.170**	**-15.12**
Stack	50	98.0	0.188	-14.77	**100.0**	0.191	-15.49	99.5	0.174	-16.58	99.5	**0.200**	-15.27	99.5	0.191	**-14.60**
	100	99.0	0.195	**-15.00**	99.5	0.190	-16.08	99.5	0.174	-16.52	**100.0**	**0.197**	-15.40	99.5	0.187	-15.15
	200	**100.0**	0.187	-15.33	**100.0**	0.190	-15.52	99.0	0.172	-16.43	**100.0**	0.192	-15.50	**100.0**	**0.198**	**-15.28**
Square	50	86.5	0.120	-15.99	79.0	0.106	-17.32	86.0	0.108	-18.04	81.5	0.114	-16.44	**89.0**	**0.125**	**-15.77**
	100	85.5	0.119	-16.38	78.5	0.103	-17.68	82.5	0.102	-18.16	81.5	0.112	-16.73	**87.5**	**0.122**	**-16.17**
	200	82.5	0.109	-16.96	75.0	0.097	-17.41	81.0	0.098	-18.11	79.0	0.107	-17.05	**88.0**	**0.119**	**-16.50**
Mug	50	63.5	0.020	-19.50	61.0	0.015	-20.93	61.5	0.016	-20.73	51.0	0.007	-19.70	**70.0**	**0.027**	**-19.24**
	100	55.0	0.011	-19.79	62.0	0.017	-20.82	57.5	0.011	-20.72	53.5	0.009	-19.84	**71.5**	**0.029**	**-19.43**
	200	59.0	0.015	-19.85	65.0	0.020	-20.24	57.0	0.011	-20.51	56.0	0.012	-19.84	**67.5**	**0.024**	**-19.53**

Averaged across the five tasks, RRTC achieves the highest SR, TPR, and LDLJ at all tested delays. Its average SR remains stable at 90.6%, 90.8%, and 90.3% for delays of 50, 100, and 200 ms, respectively. RRTC also maintains the highest average TPR (0.187, 0.183, and 0.182) and LDLJ (-15.46, -15.69, and -15.80) across the three delay settings. In contrast, the strongest baseline SR decreases from 88.5% at 50 ms to 86.7% at 200 ms. These results indicate that RRTC is insensitive to the particular inference-latency setting and remains effective as chunk generation is delayed.

At the task level, the advantage is most pronounced on Square and Mug. At 200 ms, RRTC achieves an SR of 88.0% on Square, compared with 82.5% for the strongest baseline, and 67.5% on Mug, compared with 65.0%. Similar trends are observed at shorter delays. On Lift and Stack, success rates are already close to saturation for multiple methods, leaving less room for improvement in SR. Although individual baselines occasionally perform better on specific task–metric combinations, RRTC provides the strongest aggregate performance across tasks throughout the full range of inference delays.

D.4.3 Detailed Results of Ablation Study

Table 7 reports the complete per-task ablation results. For each variant, we modify only the corresponding component while keeping all other model, training, and execution settings unchanged. Overall, the full RRTC model achieves the best performance across all three metrics on every task.

Without Re-anchoring. To ablate re-anchoring, we directly use the overlapping relative actions from the preceding chunk $R_{t-h,t\to\tau} = X_{t-h,t}^{-1} X_{t-h,\tau}$ as the prefix, without transforming them according to the current reached anchor during inference. This variant therefore retains prefix conditioning but does not compensate for the tracking-error-induced change of action anchor. Removing re-anchoring causes the largest degradation among the component ablations, reducing the average SR from 90.3% to 85.3% and LDLJ from -15.80 to -16.88. The effect is especially pronounced on Mug, where SR drops from 67.5% to 53.5%. These results confirm that re-anchoring is critical for preserving consistent absolute targets across consecutive chunks when the reached pose deviates from the previously planned pose.

Without Prefix Conditioning. We still construct the re-anchored prefix but remove its conditioning during action generation. Specifically, the prefix is no longer kept fixed during denoising; instead, all action tokens are initialized from noise and denoised jointly. This reduces the average SR to 87.0% and TPR to 0.170. The degradation is particularly large on Square ($88.0\% \to 80.5\%$) and Mug ($67.5\% \to 62.0\%$). These results show that re-anchoring alone does not enforce the corrected trajectory during generation; preserving the re-anchored actions as a fixed prefix is important for transferring the preceding chunk's trajectory constraint to the newly generated chunk.

Table 7: Ablation results on five simulation tasks at the nominal execution speed ($1\times$). We report success rate (**SR**, %), throughput-with-regret (**TPR**), and log dimensionless jerk (**LDLJ**). Higher is better for all metrics. "Average" denotes the unweighted mean across the five tasks. Best results under each task are shown in **bold**.

Task	Variant	SR↑	TPR↑	LDLJ↑
Lift	Full model	**99.0**	**0.401**	**-12.56**
	w/o Re-anchoring	97.5	0.390	-13.30
	w/o Prefix Conditioning	98.0	0.392	-12.91
	w/o Tracking-error Injection	95.5	0.381	-13.00
	w/o Pseudo-error Training	96.0	0.380	-13.02
	TT-RTC + Re-anchoring	95.0	0.357	-14.33
	REMAC + Re-anchoring	98.0	0.391	-12.74
Can	Full model	**97.0**	**0.170**	**-15.12**
	w/o Re-anchoring	95.5	0.161	-16.28
	w/o Prefix Conditioning	96.5	0.163	-15.89
	w/o Tracking-error Injection	96.0	0.160	-15.21
	w/o Pseudo-error Training	**97.0**	0.164	-15.82
	TT-RTC + Re-anchoring	95.0	0.165	-15.42
	REMAC + Re-anchoring	96.5	**0.170**	-15.96
Stack	Full model	**100.0**	**0.198**	**-15.28**
	w/o Re-anchoring	97.0	0.182	-16.43
	w/o Prefix Conditioning	98.0	0.184	-15.36
	w/o Tracking-error Injection	99.5	0.185	-15.38
	w/o Pseudo-error Training	98.5	0.178	-15.50
	TT-RTC + Re-anchoring	98.5	0.173	-16.51
	REMAC + Re-anchoring	96.5	0.184	-15.68
Square	Full model	**88.0**	**0.119**	**-16.50**
	w/o Re-anchoring	83.0	0.113	-17.37
	w/o Prefix Conditioning	80.5	0.096	-17.22
	w/o Tracking-error Injection	83.0	0.112	-16.42
	w/o Pseudo-error Training	79.0	0.102	-16.83
	TT-RTC + Re-anchoring	79.5	0.106	-18.05
	REMAC + Re-anchoring	76.0	0.101	-17.20
Mug	Full model	**67.5**	**0.024**	**-19.53**
	w/o Re-anchoring	53.5	0.009	-21.04
	w/o Prefix Conditioning	62.0	0.015	-19.73
	w/o Tracking-error Injection	63.5	0.020	-19.99
	w/o Pseudo-error Training	61.5	0.017	-19.62
	TT-RTC + Re-anchoring	60.0	0.022	-20.58
	REMAC + Re-anchoring	54.0	0.010	-19.98
Average	Full model	**90.3**	**0.182**	**-15.80**
	w/o Re-anchoring	85.3	0.171	-16.88
	w/o Prefix Conditioning	87.0	0.170	-16.22
	w/o Tracking-error Injection	87.5	0.172	-16.00
	w/o Pseudo-error Training	86.4	0.168	-16.16
	TT-RTC + Re-anchoring	85.6	0.165	-16.98
	REMAC + Re-anchoring	84.2	0.171	-16.31

Without Tracking-Error Injection. To isolate the effect of tracking-error conditioning, we disable the tracking-error injection to the policy while retaining re-anchoring, prefix conditioning, and pseudo-error training. Without this condition, the average SR decreases from 90.3% to 87.5% and TPR from 0.182 to 0.172. The degradation is again more visible on the more challenging tasks, with SR dropping from 88.0% to 83.0% on Square and from 67.5% to 63.5% on Mug. This demonstrates that explicitly providing the current tracking error helps the policy adapt the generated continuation to the reached execution state.

Without Training with Pseudo Errors. For this ablation, we retain the tracking-error-conditioned policy architecture but remove pseudo-error sampling during training. Instead of synthesizing diverse tracking-error conditions from demonstration trajectories, the policy is trained only with the identity tracking-error condition. At deployment, it is evaluated under the same controller tracking errors as the full model. Removing pseudo-error training reduces the average SR to 86.4% and yields the lowest average TPR of 0.168. The degradation is particularly evident on Square and Mug, while the SR on Can remains unchanged. This result highlights the importance of exposing the policy to diverse tracking-error conditions during training when deployment-time commanded–reached pose pairs are absent from robot-free demonstrations.

Adding Re-anchoring to Baselines. We further equip TT-RTC and REMAC with the same re-anchoring operation while keeping their original training and generation procedures otherwise unchanged. TT-RTC + Re-anchoring reaches an average SR of 85.6%, TPR of 0.165, and LDLJ of -16.98, remaining below the full RRTC on all three metrics. Similarly, REMAC + Re-anchoring reaches 84.2% SR, 0.171 TPR, and -16.31 LDLJ. These results show that correcting the action anchor alone is not sufficient to achieve the full benefit of RRTC. The policy must also be trained to operate on the resulting re-anchored prefixes and explicitly account for the current tracking error during action generation.

Furthermore, the *w/o Tracking-Error Injection* variant provides a complementary comparison that helps separate these effects. This variant can be viewed as TT-RTC augmented with re-anchoring during both training and inference. Compared with TT-RTC with re-anchoring, this variant therefore additionally adapts the policy to re-anchored prefix distributions during training. Nevertheless, removing tracking-error injection still reduces the average SR from 90.3% to 87.5%. This shows that learning to generate from re-anchored prefixes alone is not sufficient. The tracking-error injection module offers additional information that helps the policy adapt to different tracking errors.

E Details of Real-World Evaluation

E.1 Task Descriptions

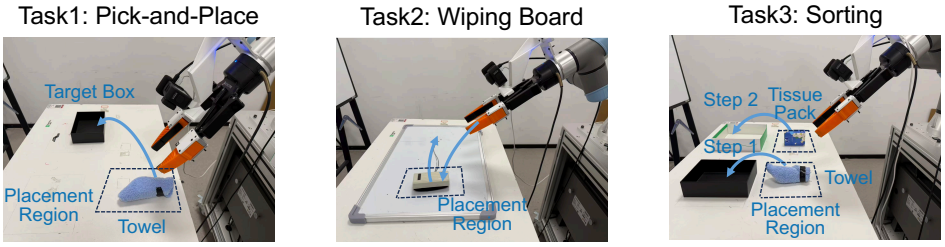


Figure 12: **Real-world task setups.** (a) Towel Pick-and-Place: the towel is initialized within the indicated placement region and transferred to the target box. (b) Board Wiping: the cleaning tool is initialized within the indicated region and used to wipe the marked line. (c) Tabletop Sorting: the robot sequentially places the towel and tissue pack into their corresponding target boxes.

We evaluate RRTC on three real-world manipulation tasks with complementary execution characteristics. For each task, we collect 100 robot-free demonstration episodes and evaluate each method over 30 physical-robot rollouts. Across methods, we use the same task setup and randomization protocol. Execution time is measured from the issuance of the first robot command until the task success criterion is first satisfied. A human evaluator terminates the rollout once the predefined success condition is met. Failed rollouts continue until the task-specific maximum duration; they are counted when computing SR but excluded from the average completion time.

Towel Pick-and-Place. The robot is required to grasp a towel from the tabletop, transport it to a designated target box, and release it. This task involves deformable-object manipulation, for which the changing object geometry introduces additional spatial and temporal complexity (Lin et al., 2022). In our setting, the towel shape may vary across episodes. At the beginning of each rollout, the towel position and orientation are randomized within a placement region, as shown in Figure 12(a).

A rollout is considered successful if the robot successfully grasps the towel, transports it to the target region, and releases it such that the towel remains within the box. This task primarily evaluates whether the policy can maintain reliable and smooth execution while manipulating a deformable object. The maximum rollout duration is 20 s. Typical failure modes include missed grasps caused by abrupt motion around chunk handovers, and inaccurate placement that leaves the towel outside the target box.

Board Wiping. The robot is required to wipe a marked line on a whiteboard using a cleaning tool held by the gripper. Successful execution requires the end effector to establish and maintain contact with the board throughout the wiping motion. Contact-rich manipulation is particularly sensitive to pose uncertainty and controller feedback (Ranjbar et al., 2021). Accordingly, abrupt changes in the commanded trajectory can disrupt contact and reduce wiping effectiveness. The initial tool placement is randomized, as shown in Figure 12(b). A rollout is considered successful if the marked line is wiped clean. This contact-rich task places stronger demands on motion continuity, as abrupt changes in the commanded trajectory can disrupt contact and reduce wiping effectiveness. The maximum rollout duration is 20 s. Typical failure modes include failure to grasp the cleaning tool and loss of contact with the board during wiping, often following abrupt motion or jitter around chunk transitions.

Tabletop Sorting. The robot is required to sequentially pick up two objects, a towel and a tissue pack, from the table and place them into their corresponding target boxes, as shown in Figure 12(c). The initial positions of the objects are randomized. The task consists of multiple consecutive manipulation stages. Coordinating the transitions between such stages is a central challenge in long-horizon manipulation (Chen et al., 2023), placing greater demands on both action continuity and execution reliability. A rollout is considered successful if both objects are placed into their corresponding target boxes. This task evaluates real-time execution over a longer horizon with more action-chunk transitions. The maximum rollout duration is 50 s. Typical failure modes include missed grasps and abrupt motion that drives the end effector into unseen, out-of-distribution states, from which the policy may fail to recover.

E.2 Detailed Setups

We use $\pi_{0.5}$ as the real-robot policy backbone with a single valid wrist-camera input and an action horizon of $H = 32$. For each task, we first adapt the pretrained $\pi_{0.5}$ checkpoint to obtain a task-specific Vanilla policy. The vision-language backbone is frozen, while the complete action expert and the action- and timestep-projection layers are fine-tuned for 10000 steps using AdamW (Loshchilov & Hutter, 2017) and a cosine learning-rate schedule (Loshchilov & Hutter, 2016) from 5×10^{-6} to 5×10^{-7}.

All baselines and RRTC are initialized from the same Vanilla checkpoint, fine-tuned for 2,000 steps, and trained using the same demonstrations, optimizer, learning-rate schedule, and adaptation budget. We use a prediction horizon of $H = 32$ and an execution horizon of $h = 8$, while limiting the temporal offset used for RRTC training to $\kappa_{\max} = 4$.

The policy operates at 30 Hz, corresponding to a control interval of $\Delta t = 33.3$ ms. End-to-end policy inference takes approximately 68–75 ms, while communication introduces an additional latency of approximately 112–180 ms, resulting in a total system latency of 180–255 ms. This latency corresponds to a delay of $d = 5$–8 control steps; therefore, for TT-RTC and RRTC, we set the maximum action-prefix length $L_p^{max} = 8$ to accommodate the worst-case delay.

For low-level Cartesian control, we use the `servoL` controller provided by the UR robot, with `gain`=400, a control period of 2 ms, and `lookahead_time`=0.1 s. To evaluate tracking accuracy, we replay each task demonstration at its original execution speed and record the translational and rotational tracking errors at every control step. We aggregate the recorded errors across all replayed demonstrations and tasks. The resulting translational tracking error is 10.14 ± 5.44 mm, and the rotational tracking error is $0.822 \pm 0.478^\circ$, where the values denote the mean and standard deviation.