SAIL: Faster-than-Demonstration Execution of Imitation Learning Policies

Nadun Ranawaka Arachchige[1*], Zhenyang Chen[1*], Wonsuhk Jung[1], Woo Chul Shin[1], Rohan Bansal[1], Yu Hang He[1], Yingyan Celine Lin[1], Benjamin Joffe[1,2], Shreyas Kousik[1†], Danfei Xu[1†]

***Abstract*—Offline Imitation Learning (IL) methods such as Behavior Cloning are effective at acquiring complex robotic manipulation skills. However, existing IL-trained policies are confined to executing the task at the same speed as shown in demonstration data. This limits the *task throughput* of a robotic system, a critical requirement for applications such as industrial automation. In this paper, we introduce and formalize the novel problem of enabling faster-than-demonstration execution of visuomotor policies and identify fundamental challenges in robot dynamics and state-action distribution shifts. We instantiate the key insights as SAIL (Speed Adaptation for Imitation Learning), a *full-stack system* integrating four tightly-connected components: (1) a consistency-preserving action inference algorithm for smooth motion at high speed, (2) high-fidelity tracking of controller-invariant motion targets, (3) adaptive speed modulation that dynamically adjusts execution speed based on motion complexity, and (4) action scheduling to handle real-world system latencies. Experiments on 12 tasks across simulation and two real, distinct robot platforms show that SAIL achieves up to a 4× speedup over demonstration speed in simulation and up to 3.2× speedup in the real world. Additional detail is available at https://sail-robot.github.io.**

I. Introduction

Speed is essential for the real-world application of robot learning. High-speed industrial robots automation of production lines has been the driving force of the modern industrial revolution. Towards enabling such automation benefits outside of production settings, recent offline imitation learning methods [7, 35] have been tremendously successful on tasks that traditional systems cannot solve, such as manipulating deformable objects or performing non-prehensile manipulation. The key to these methods is to learn from high-quality human task demonstrations. However, humans tend to demonstrate slowly for complicated manipulation tasks. While recent works [8, 32, 35] have made collecting high-quality data easier, slow demonstrations are still inevitable due to user inexperience or natural limitations in human sensorimotor capabilities. Thus, this paper seeks to answer the question: *How can we speed up the execution of learned visuomotor policies beyond the original demonstration speed?*

The central challenge to speeding up is that changing the execution speed of a policy alters the robot's dynamic response to the predicted actions. In particular, executing the same action at different speeds can result in varying degrees of tracking error and dynamic effects depending on the underlying controller implementation. This in turn shifts the observation distribution, which further causes the policy to deviate from its prior distribution. For these reasons, the few existing methods that attempt to speed up imitation learning require the human operator to demonstrate multiple speeds [27, 25] and rely on subsequent self-supervised online learning with task-specific reward design [26], enabling a 1.1–1.3× increase in execution speed, at the expense of considerable loss in success rate. Speeding up is also limited by inference speed, sensor delays, and control bandwidth [27], making it a full-stack challenge. As a result, most prior studies on offline imitation learning inherently assume that the underlying robotic system have identical behaviors during data collection time and policy rollout, including the execution speed.

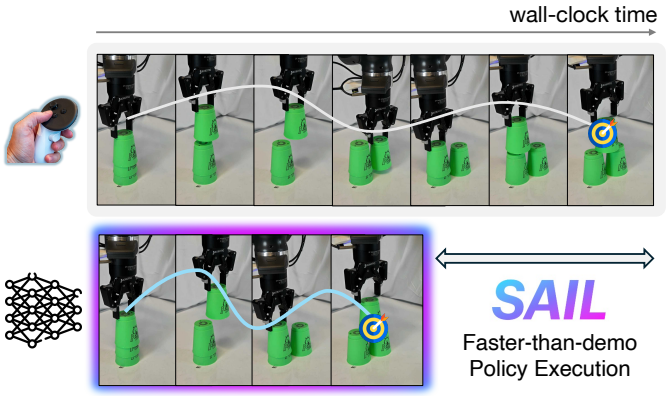


Fig. 1: The goal of our system, Speed-Adaptive Imitation Learning (SAIL), is to speed up the execution of a learned visuomotor policy such that the robot can complete manipulation tasks faster than shown in the original training demonstrations while maintaining task success.

Our key insight is that faster-than-demonstration execution has two fundamentally coupled requirements: (1) high-fidelity action tracking under varying time parametrization and (2) temporally-consistent policy predictions to prevent trajectory discontinuities. High-fidelity tracking enables the robot to faithfully execute predicted motion at higher speeds by minimizing tracking error and compensating for dynamic effects. However, high-gain control also amplifies any inconsistencies in the predicted actions across consecutive prediction steps in receding horizon control. These inconsistencies arise from the distributional shift in robot behavior at higher speeds, creating a vicious cycle: shifted dynamics lead to out-of-distribution observations, causing incorrect and inconsistent

∗ Equal contribution, † equal advising
[1]Georgia Institute of Technology
[2]Georgia Tech Research Institute

predictions that result in discontinuous reference trajectories and jerky robot motion, which in turn create even more distributional shift. Such prediction inconsistencies exist even when executing at demonstration speeds [17], but they typically go unnoticed in prior work where low-gain controllers inadvertently smooth out these inconsistencies at the cost of precise tracking. We empirically validate this phenomena in a controlled experiment (Sec. VII-G2).

SAIL addresses this coupling challenge through two core components:

- *High-fidelity action tracking.* We employ a high-fidelity tracking controller that minimizes tracking error even at increased speeds. A key insight is that rather than trying to follow commanded poses from demonstrations, which can lead to unstable behavior when sped up, we train our policy to predict the actual poses that the robot achieved during demonstrations. Combined with high-gain feedback and feedforward velocity terms, this allows the robot to accurately track trajectories at faster-than-demonstration execution speeds.
- *Consistency-preserving trajectory generation.* We employ Classifier-Free Guidance (CFG) [12] with Diffusion Policy [7] to ensure temporal coherence during receding horizon execution by conditioning each new trajectory prediction on previously executed actions. When tracking error remains low, this approach maintains consistency between consecutive policy outputs, preventing discontinuities that could cause jerky motion at high speeds. When significant tracking errors occur, the system automatically reverts to unconditioned predictions to maintain responsiveness to real-time feedback.

We further complement this core approach with two practical components that are essential for real-world deployment of high-speed visuomotor policy.

- *Adaptive speed modulation.* This component automatically adjusts execution speed based on the complexity of the current motion. It slows down during fine-grained manipulation phases where physical limitations necessitate precise control, such as insertion or grasping tasks, while maintaining high speeds during coarse movements.
- *Action scheduling.* To maintain real-time performance at high speeds, this component handles various system latencies by carefully scheduling each event in the control loop such as network inference and sending commands to the robot controller. It ensures proper synchronization between observations and actions, preventing potential out-of-distribution policy input from misaligned signals.

This full-stack approach allows SAIL to maintain high task success rates while achieving significantly faster-than-demonstration execution speeds across a variety of manipulation tasks. We empirically validate our key technical insights through controlled experiments in simulation, where we systematically evaluate the impact of controller gains, prediction consistency, and speed modulation on task performance. We demonstrate that SAIL achieves up to a $4\times$ speedup over demonstration speed while maintaining high task success rates. We further validate SAIL on physical robot systems, where it achieves up to a $3.2\times$ speedup across challenging tasks including cup stacking, oven baking, contact-rich wiping, and cloth folding.

II. Preliminaries, Challenges, and Problem Statement

In this section, we first outline our policy and controller hierarchy and describe the context and form of the imitation learning policy used in this work. Then, we explain the unique challenges that result from attempting to accelerate policy execution and state the core research problem of this work.

A. Policy and Controller Hierarchy

We consider a robot control system that consists of two levels: (1) a high-level neural network policy $\pi(x,o)$ that generates an action command a given the current robot state x and sensory observation o, and (2) a low-level robot controller $\mathcal{K}$ that translates the action to robot joint torque. For instance, given an end effector-space controller, we assume a learned policy model that can generate a predicted action trajectory $a_t = [x_t^{\mathrm{d}}, x_{t+1}^{\mathrm{d}}, \cdots, x_{t+H-1}^{\mathrm{d}}]$, where H is the prediction horizon, and each x_i^{d} includes a desired SE(3) end effector pose and the corresponding gripper open/close command at timestep i. The controller then tracks this trajectory by calculating the low-level joint torque command u_t given each desired end effector pose and the current state x_t as

$$u_t = \mathcal{K}(x_t^{\mathrm{d}}, x_t, \delta^*), \tag{1}$$

where δ^* is the time interval between discrete configurations in the reference trajectory x^{d}.

The robot controller $\mathcal{K}$ typically runs at a much higher rate than both policy model inference *and* the time interval of the policy's output actions. For example, a typical torque-based controller may run at 500 Hz, while policy inference runs at 10 Hz and outputs predicted configurations x_t^{d} with time interval $\delta^* = 0.05$ s (i.e., 20 Hz). This meaning that each predicted configuration x_t^{d} is tracked for 25 controller steps before the controller moves on to the next step. The policy is usually executed in a receding-horizon manner, where each reference trajectory is executed for $H^{\mathrm{e}} < H$ steps before a new reference trajectories is generated by the policy. So, in the example above, 2 policy configurations would be tracked before rerunning inference is complete.

B. Offline Imitation Learning Context and Challenges

We consider the setting of offline imitation learning where we aim to learn the policy π from a dataset of n demonstrations $\mathcal{D} = (o_t, x_t, x_t^{\mathrm{d}})_{t=1}^{n}$ collected through teleoperation, where each datapoint consists of the sensory observation o_t, robot state x_t, and controller-specified desired configuration x_t^{d} sampled at fixed time interval δ^*. For example, x^{d} can be the task-space target pose set by a teleoperation device. An action supervision a_t is constructed by extracting length-H desired configurations $[x_t^{\mathrm{d}}, x_{t+1}^{\mathrm{d}}, \cdots, x_{t+H-1}^{\mathrm{d}}]$ from the demonstration sequence.

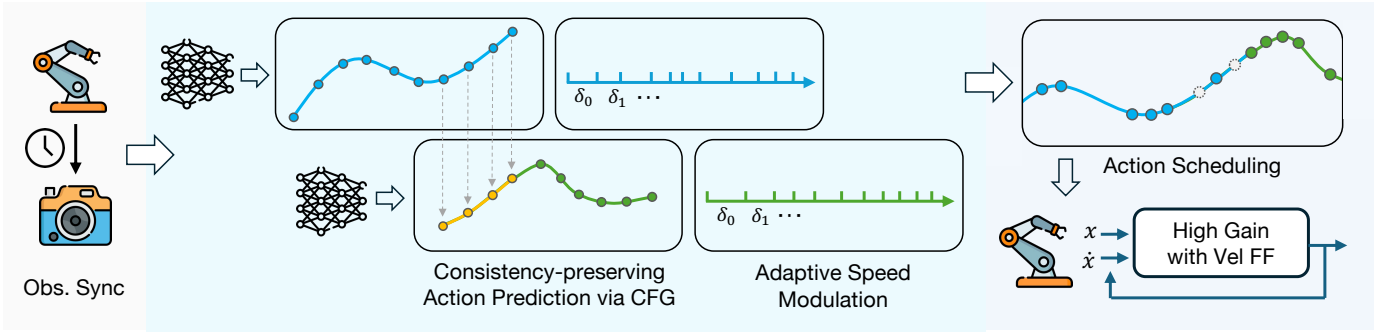


Fig. 2: **SAIL System Overview.** Our framework operates at two levels: (a) **Policy Level**: Starting with synchronized observations (Obs. sync) from robot state and camera inputs, the system generates (1) temporally-consistent action predictions through action-conditioned CFG and (2) time-varying action interval δ_t. (b) **Controller Level**: The predicted actions are scheduled for execution while accounting for sensing-inference delays, with outdated actions being discarded. The scheduled actions are executed using a high-gain controller with velocity feedforward (Vel FF) terms to track trajectory at the specified time parametrization.

A key challenge in offline imitation learning is distribution shift caused by compound error [24]; when the policy makes errors in action prediction during execution, it encounters states that differ from those in the training data, which in turn causes higher prediction errors. This challenge is particularly acute when we attempt faster-than-demonstration execution ($\delta < \delta^*$), as both the dynamics of the system and the distribution of tracking errors change in ways not captured in the training data.

C. Policy Model

As a representative high-performance behavior cloning method, we consider Diffusion Policy (DP) [7] as our main policy model. DP can generate multi-step trajectories through an iterative denoising process. Given the current observation and state (o_t, x_t), the policy generates the desired trajectory a by progressively denoising from random noise through N steps:

$$a_n = a_{n+1} - \gamma \mu_\theta(o_t, x_t, a_{n+1}, n) + \mathcal{N}(0, \sigma_n^2 \mathbf{I}) \tag{2}$$

where μ_θ is a learned denoising model and σ_n controls the noise schedule.

This architecture both offers advantages and introduces challenges. The key advantage is that the underlying diffusion model can generate action sequences rather than single-step actions, also known as action chunking [7, 35]. This makes it possible to execute a policy where it takes longer time to infer an action than to execute one. This property is especially important when speeding up policy execution, as the network inference speed loop is inherently limited. However, the probabilistic nature of the model makes it prone to predicting inconsistent trajectories across two consecutive observations [17], causing jerky robot motion under high speed. Moreover, the multi-step denoising process introduces additional latency that must be carefully managed when executing motions faster than demonstrations. These challenges motivate our core components, most notably action inpainting for consistent prediction (subsection III-B) and compensation for system latency and inference delays to ensure real-time operation (subsection III-D)

D. Adaptive-Speed Policy Execution Challenges

We identify two key challenges for adaptive-speed policy execution. First, increasing the speed of policy execution means that the controller $\mathcal{K}$ must be capable of tracking trajectories at higher speed than in demonstration. As noted earlier, for learned policies, it is common to assume each action is executed with a fixed time interval δ^*, and thus a fixed number of control steps for the controller $\mathcal{K}$ to attempt to reach each desired configuration. This time interval is usually determined by the training data and is the same for recording during data collection and for the execution of predicted trajectories during deployment. Therefore, changing this time interval *during* deployment requires the robot's tracking controller to be capable of reaching some poses more quickly. Second, not all parts of tasks are suitable for acceleration (e.g., manipulation steps such as alignment before grasping require precise motion and fine-grained adjustment), so the robot must be able to adaptively modulate its speed based on task progress. In subsection III-C, we develop a method to enable the policy model to adjust the timing of the predicted action trajectory based on the complexity of the predicted motion.

E. Problem Statement

To summarize, we seek to solve the following. Given a policy trained with action data logged at a constant time interval δ^* (e.g., 0.1s for a 10 Hz recording frequency), our goal is to execute the policy with a different time parametrization $\delta_t = c_t \delta^*$, where $c_t \in (0, 1]$ is the *speedup factor* that determines execution acceleration at time t. We seek to be able to vary c_t while ensuring that task success does not decrease with respect to the baseline policy (i.e., $c_t = 1$). Note that this speedup factor notation directly reflects the practical implementation of speeding up imitation learning, because,

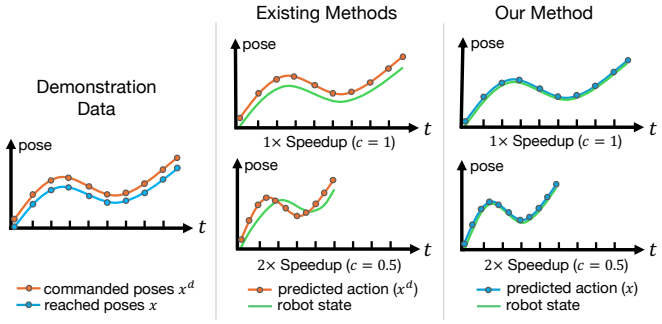


Fig. 3: **Commanded vs Reached Pose.** (Left) During teleoperated data collection, the operator commands the robot to reach pose x^{d}. The robot controller (often low-gain to minimize unintended motion) incurs tracking error and reaches poses x. (Middle) Most existing methods train policies to predict x^{d}, assuming similar controller errors during rollout. However, speeding up policy execution changes the error profile, causing state distribution shift. (Right) To mitigate the error shift at higher execution speed, we propose to train policy model to predict the reached poses x in demonstration and track the predicted actions with a high-fidelity controller.

in practice, we directly change the time parameterization, as opposed to the robot speed.

III. Speed-Adaptive Imitation Learning (SAIL)

The goal of our speed-adaptive policy execution is for a low-level robot controller to generate high-speed motion to track a reference trajectory produced by the learned policy while minimizing the induced compound shift in observation and action distributions. To address these problems, we propose several key design choices for SAIL: a high-fidelity tracking controller, an approach to ensure consistency between receding-horizon policy predictions, adaptive speed detection, and a strategy for handling system latency. We illustrate the system in Figure 2.

A. High-fidelity Tracking Control

One key challenge of executing policy faster with a new time parametrization $\delta_t < \delta^*$ is the change in controller behavior, which we find can lead to different tracking error profiles unseen from the training dataset. This error shift leads to distribution shift and compounding error in policy rollout. This section aims to mitigate this controller error shift.

Existing Methods: Assume Fixed Controller Behavior. During teleoperated data collection, the operator commands the robot to reach pose x^{d}, while the robot controller incurs tracking error and reaches poses x. This error $e = x^{\mathrm{d}} - x$ is usually nonnegligible, as it is common to use low-gain compliant controller to minimize noisy actions from unintended operator motion. Typically, policy models are trained to predict the commanded poses x_t^{d}. In this case, the same controller $\mathcal{K}$ and time parametrization δ^* are used to generate torque commands $u_t = \mathcal{K}(x_t^{\mathrm{d}}, x_t, \delta^*)$ during both demo collection and policy rollout. In other words, the controller behaviors and tracking errors are assumed to be consistent between training and execution.

Challenge: Controller Behavior Shift at Higher Speed. However, if we use the same controller $\mathcal{K}$ for speeding up execution (δ instead of δ^*), we would have different controller behavior $\hat{u}_t = \mathcal{K}(x_t^{\mathrm{d}}, x_t, \delta)$. Such behavior changes during execution lead to a new error profile $\hat{e}$.

High-fidelity Tracking of *Reached* Configurations. Designing a controller to match e in the demonstration under higher execution speed is challenging. Instead, we tackle this problem by recognizing that the training data includes the *reached* configuration x of the robot given every action. Therefore, by tracking x instead of x^{d}, we bypass the problem of matching controller behavior and error e. We illustrate this difference in Figure 3. This strategy requires high-fidelity tracking to ensure the robot can actually reach x. We propose to use a high-gain controller plus a feedforward velocity term to track x as closely as the robot physically allows. We include more details of controller design in Appendix Sec. VII-K.

B. Ensuring Consistency Between Policy Predictions

The next key challenge we seek to address is *divergence* during receding horizon execution, as shown in Figure 4. We found empirically that, when running receding-horizon inference, our policy would occasionally predict sudden large changes in the reference trajectory. These divergence events are sometimes task-relevant (i.e., tracking a diverging trajectory will lead to task success), and other times erroneous due to noisy or out-of-distribution observations. We use two key strategies to handle this challenge. First, to improve consistency between predictions, we use Classifier-Free Guidance (CFG) [12] via conditioning on previously-generated trajectories. Second, we correlate task-relevant divergence with the robot's *tracking error* to establish a cutoff that lets us autonomously decide whether or not to trust a new prediction. That is, whether to *avoid* or *exploit* divergence.

We assume that we have actions $a_{0:H}$ generated by Diffusion Policy [7]. From this sequence, we execute the first H^{e} actions, i.e., $a_{0:H^{\mathrm{e}}}$. Then we execute next H^{f} actions, i.e., $a_{H^{\mathrm{e}}:H^{\mathrm{e}}+H^{\mathrm{f}}}$, *while* generating the next actions $\hat{a}_{0:H}$ to execute.

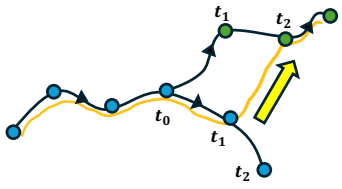


Fig. 4: **Divergence during receding horizon execution**. We found that the diffusion policy would occasionally produce diverging or inconsistent predictions between receding-horizon planning iterations. For example, the blue and green trajectories are two consecutive trajectories that diverge in path. With sensing and inference delay, this divergence causes jerky movement (yellow line) with receding horizon control.

After generation, we start to execute $\hat{a}_{H^{\mathrm{f}}:H^{\mathrm{f}}+H^{\mathrm{e}}}$ and repeat this process.

Classifier-Free Guidance (CFG) for Action Consistency. Our objective is to ensure that the initial part of the second predicted sequences $\hat{a}_{0:H^{\mathrm{f}}}$ remains consistent with the actions actually executed from the previous prediction, $a_{H^{\mathrm{e}}:H^{\mathrm{e}}+H^{\mathrm{f}}}$. Mathematically, we aim to sample from the conditional probability distribution: $p(a_{0:H}|o_0, a_{0:H^{\mathrm{f}}})$. To train the model, we restructure the data as follows. Given a datum $(o_t, a_{t:t+H})$ from the dataset $\mathcal{D}$, we denote the full action sequence $a^{\mathrm{f}} = a_{t:t+H}$ and the predicted actions for conditioning as $a^{\mathrm{c}} = a_{t:t+H^{\mathrm{f}}}$. Following the standard CFG algorithm, we train both a conditional score function $\varepsilon_\theta(a^{\mathrm{f}}, a^{\mathrm{c}})$ and an unconditional score function $\varepsilon_\theta(a^{\mathrm{f}}, \phi)$, where ϕ represents null actions.

For inference at runtime, given a current observation o_t and action conditioning $a^{\mathrm{c}} = a_{H^{\mathrm{e}}:H^{\mathrm{e}}+H^{\mathrm{f}}}$ (which is taken from the policy prediction from the previous receding horizon iteration), we perform the diffusion process using a score function obtained as

$$\varepsilon_\theta(a^{\mathrm{f}}, a^{\mathrm{c}}) = \varepsilon_\theta(a^{\mathrm{f}}, a^{\mathrm{c}}) + w(\varepsilon_\theta(a^{\mathrm{f}}, a^{\mathrm{c}}) - \varepsilon_\theta(a^{\mathrm{f}}, \phi)), \quad (3)$$

where w is the CFG guidance weight. For our case, w becomes a parameter that balances the fidelity between current observation and consistency to previous actions. For further details, we refer the reader to [12].

Tracking Error Cutoff for Exploiting Divergence. Through empirical study (see the experiments Sec. IV and in Appendix subsection VII-I), we observed that CFG performs well when the action conditioning aligns with the *unconditional* action distribution (i.e., the output of the policy with no action conditioning). However, when speed increases, the action conditioning often falls outside the unconditional action distribution, leading to performance degradation. We identified tracking error as a key factor influencing this issue. To decide when to exploit divergence, we then evaluate whether a distance metric between the current and desired states exceeds a predefined threshold. If the error exceeds a threshold ρ, we apply CFG guidance; otherwise, we do not. Despite its simplicity, we found in practice that this strategy is effective.

C. Adaptive Speed Modulation

In the context of robotic manipulation tasks, not all actions can be executed at the same high speed without compromising task success. We call these *critical actions*, which require precise, often nonlinear movements and interactions, such as grasping a delicate object or maneuvering into a tight space. Critical actions typically require a slower execution speed to maintain accuracy and reliability, as we show in subsection IV-B. To address this, we design an *adaptive speed modulation* so that the robot can increase the speed up factor c (slow down) to perform critical actions during task execution. Specifically, we aim to identify critical actions in demonstrations, train the policy to predict both the actions and their critical action labels, and dynamically the speed up factor c accordingly during execution.

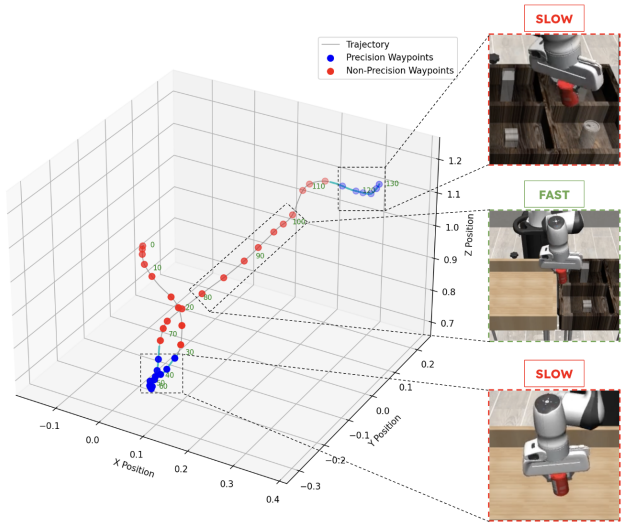


Fig. 5: **Policy rollout with adaptive speed modulation** . Waypoints (red) are generated by [29] given the trajectory as input along with an error threshold. Areas of complex motion (blue), are marked by performing a spatial clustering of the extracted waypoints. Any waypoint outside the cluster threshold we label as noise. Frame numbers are labeled every 10 steps in green – one can observe that the clustering has been performed properly by the increased concentration of frame numbers in clustered areas (the end effector spends more time in these regions).

We propose two techniques to identify critical actions from the demonstration data: measuring motion complexity and using gripper open/close actions (see Appendix subsection VII-F. Each technique returns a binary *critical action flag* $k_t \in \{0,1\}$ ($k_t = 1$ means that a_t is critical), which we use to set the speedup factor c_t. Given the binary critical action flag k_t, we set the speedup factor as:

$$c_t = k_t \cdot c^{\mathrm{slow}} + (1 - k_t) \cdot c^{\mathrm{fast}}. \quad (4)$$

where $c^{\mathrm{slow}} \in (0,1]$ is a slower speedup factor used for critical actions and $c^{\mathrm{fast}} \in (0,1]$ is used otherwise. Note that $c^{\mathrm{slow}} > c^{\mathrm{fast}}$ since the *reciprocal* of the speedup factor determines the speedup. We set c^{slow} and c^{fast} to empirically-validated presets for each task. Also note that we lower-bound c_t next in subsection III-D. A policy rollout using this technique of adaptive speed modulation is shown in Figure 5.

D. Continuous Control Loop with System Latency

In real-world robotic systems, different sensors and actuators are driven by separate micro-controllers, with different communication protocols and sampling rates. Echoing findings in prior work [8], we observe that the latencies caused by such heterogeneity can lead to out-of-distribution inputs to the policy and time-misaligned action commands to the controller. Moreover, even with an ideal system, the maximum achievable speedup is constrained by fixed delays such as data transmission and neural network inference time. We need to derive various hyperparameters, including policy prediction horizon

and minimum speedup factor c, based on such constraints to ensure that the control loop can run continuously in real time. In the following paragraphs, we will discuss three system design decisions that address these challenges.

Observation Synchronization. To address the *sensing latencies* from various sensors, we quantify the latency of the system components and create global timestamps for the camera observations and robot states. Specifically, observation timestamps are aligned to the timestamps of camera that with the highest latency, and proprioceptive observations are calculated from linear interpolation of robot states at those timestamps. During data collection, the recorded action is synchronized with the nearest camera observation timestamp.

Action Scheduling. There is an *irreducible delay* δ^{delay} between the moment that new data is requested from the sensors t^o and the corresponding action is generated t^a. To accommodate this delay, SAIL's control loop initiates the sensing and inference at t^o while continuing to execute the remaining actions from the prediction until a new action is generated at t^a. The action trajectory is then scheduled according to the set action interval δ_t (time-varying if using adaptive speed modulation) starting at t^o. The system then discards outdated actions with timestamp older than t^a and proceeds to execute the rest of the actions. This process is illustrated in Figure 6.

Lower-Bounding the Speedup Factor. The sensing and inference delay δ^{delay} limits the maximum speedup (i.e., the minimum c_t) that can be supported by the system. Therefore, we seek to configure the system such that the control loop prevents **action exhaustion**, i.e., running out of actions to be executed before the next action inference is finished. The lower bound δ^{lb} is expressed as:

$$\delta^{\text{lb}} > \frac{\delta^{\text{delay}}}{H^{\text{p}} - H^{\text{c}}}. \tag{5}$$

which in turn determines the speedup factor $c_t \in \left(\delta^{\text{lb}}/\delta^*, 1\right]$.

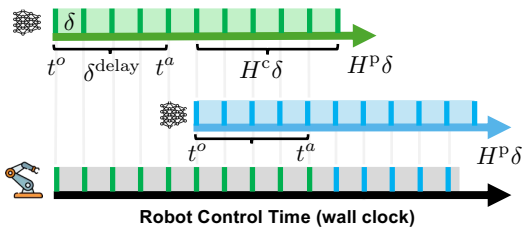


Fig. 6: **Handling Latencies in Control Loop.** We illustrate the control loop timeline of SAIL and how it handles system latency. The green timeline (top) shows the first action sequence generated at t^o. The sequence spans $H\delta$ with the last H^{c} steps for conditioning the next prediction. The blue timeline (middle) shows the next action prediction starts while the system continues to execute the first prediction. The bottom timeline shows actual robot execution timeline. The system smoothly transitions from the first sequence (green) to the next (blue) while maintaining a continuous control loop.

Detailed derivation can be found in Appendix VII-E.

IV. Evaluations

In our experiments, we aim to validate our key insights by testing each design choice. Then, we demonstrate the capability of the system to enable faster-than-demonstration-execution while maintaining task success rate relative to baselines. Our validation experiments seek to test the following hypotheses:

- ***H1***: Combining a high gain controller with predicting reached poses increases task success rate during high-speed execution.
- ***H2***: Using a high gain controller requires smooth reference trajectories for a high success rate.
- ***H3***: Consistent-preserving action prediction generates temporally-consistent actions that further improves policy performances at high speed.
- ***H4***: Adaptive speed modulation improves policy success rate a high execution speed.
- ***H5***: Our system enables faster-than-demo execution while keeping a high success rate across simulation and real robot systems.

This section is organized as follows. First, we explain our simulation experiment setup. Then, we test H4-H5 and demonstrate the main experiment results in subsection IV-B. We leave the testing of H1–H3, which are the hypotheses for different components of SAIL, to the Appendix VII-G.

A. Experiment Setup (Sim)

Tasks. We evaluate on the following tasks from the RoboMimic [19] benchmark and MimicGen [20]: Lift, Can, Square, Stack, and Mug Cleanup.

Receding Horizon Formulation. We perform all experiments except those involving demo replay (***H1***) in a *receding horizon* manner. Note that, during inference, the robot continues executing actions from the previous policy output until inference is complete, thereby simulating the inference delay.

Metrics. We consider two suites of evaluation metrics to quantify the contributions of different components in SAIL. The first suite focuses on task performance and efficiency, which includes task success rate (SR), throughput-with-regret (TPR), average time for successful rollouts (ATR), speedup-over-demo (SOD). The second suite illustrates the characteristics of the trajectory, which contains consistency (CON), spectral arc length[13] (SPARC), log dimensionless jerk[13] (LDLJ), and weighted Euclidean distance[17] (WED). More details on the metrics are presented in Appendix Sec. VII-B.

B. Testing *H4* and *H5*: SAIL Achieves High Task Throughput

Since few imitation learning methods directly address execution speedup, we design several baselines for comparison, including AWE [29] which achieves speedup as a side effect of its waypoint-based approach. Our baselines are:

- **DP** [7]: Executes actions at the original demonstration speed, serving as our primary baseline.

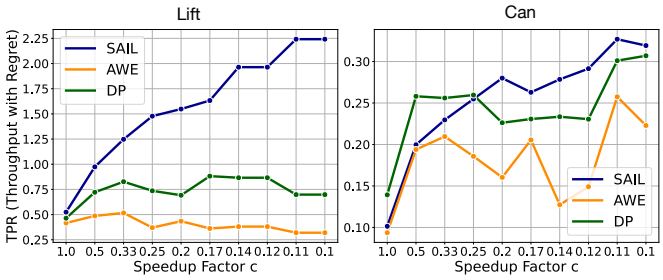


Fig. 7: **Speedup Factor vs. TPR on Can and Lift Tasks.** We show that as the speedup factor increases, SAIL's throughput-with-regret increases more than the AWE and DP baselines. In other words, SAIL is able to accumulate more task successes more quickly while limiting task failures.

- **DP-Fast**: Executes Diffusion Policy actions at an accelerated fixed frequency using a low-gain controller, representing the naive approach to speedup.
- **Aggregated Actions**: Operates in delta Cartesian Space by aggregating consecutive actions in similar directions (detailed in Appendix Sec. VII-J).
- **AWE** [29]: Uses automatically extracted waypoints to generate absolute action labels.
- **SAIL(-C)**: Ablation of our method without consistency-preserving trajectory generation.
- **SAIL(-AS)**: Ablation of our method without adaptive speed modulation.

SAIL achieves much higher throughput than baselines. As seen in Table I, SAIL can achieve up to 3× throughput of baselines such as DP [7] for the **Can** and **Stack** tasks, without sacrificing success rate. We attribute this to the combination of components including the high-gain controller, tracking reached poses, and adaptive speed modulation. Moreover, we conducted a thorough study of how the throughput (TPR) changes as we vary c_t in Figure 7. We observe that SAIL is able to smoothly improve the TPR as c_t increases, with up to $c_t = 0.1$ (10× speed) showing is robustness at least in an ideal simulated environment.

Adaptive speed modulation (slow down) is necessary for success in high-precision tasks. As seen in the results for **Square**, our ablation baseline (SAIL-AS) without the adaptive speed modulation achieves a reduced success rate when compared to SAIL (0.86 vs. 0.64).

Action Conditioning improves Success Rate. For many of our tasks, when our CFG-based action conditioning was enabled, it resulted in a higher success rate and throughput, indicating that action conditioning can help the model generate more consistent actions and achieve better performance.

V. Real-World Evaluation

We verified our method on two different robot systems - Franka and UR5, across seven challenging tasks. Despite the difference in the underlying control system, we demonstrated that our recipes are general for achieving faster-than-demonstration execution on real robots.

TABLE I: Results of evaluation in Sim

		DP [7]	DP-F	AWE [29]	Agg. Act.	BID[17]-Fast	SAIL
Lift	SR ↑	**1.00**	0.95	**1.00**	0.91	0.86	**1.00**
	TPR ↑	0.46	1.02	0.44	0.52	0.91	**1.68**
	ATR ↓	2.23	1.52	2.35	1.78	0.97	**0.61**
	SOD ↑	1.08	1.59	1.02	1.37	2.50	**3.98**
Can	SR ↑	**0.97**	0.87	0.96	0.82	0.79	0.92
	TPR ↑	0.18	0.37	0.17	0.16	0.34	**0.51**
	ATR ↓	5.52	2.34	5.80	4.77	2.39	**1.81**
	SOD ↑	1.05	2.48	1.00	1.22	2.34	**3.20**
Square	SR ↑	0.83	0.55	0.83	0.29	0.49	**0.86**
	TPR ↑	0.10	**0.15**	0.10	0.03	0.12	0.13
	ATR ↓	7.56	**3.42**	8.13	4.81	3.45	6.41
	SOD ↑	0.99	**2.20**	0.93	1.57	2.18	1.18
Stack	SR ↑	**1.00**	0.98	0.98	0.82	0.99	0.98
	TPR ↑	0.19	0.44	0.11	0.17	0.47	**0.66**
	ATR ↓	5.50	2.37	9.01	6.18	2.61	**1.56**
	SOD ↑	0.98	2.28	0.60	0.87	2.07	**3.47**
Mug	SR ↑	0.68	0.56	**0.75**	0.59	0.62	0.72
	TPR ↑	0.03	0.05	0.02	0.03	0.06	**0.08**
	ATR ↓	17.44	9.67	28.79	15.86	8.71	**8.09**
	SOD ↑	0.97	1.74	0.59	1.06	1.94	**2.09**

TABLE II: Real-World evaluation of SAIL.

		SR	TPR	ATR	SOD
Stacking Cups	DP-Fast	0.10	-2.28	**14.00**	**1.85**
	SAIL	**0.40**	**-0.12**	14.71	1.76
Wiping Board	DP-Fast	**0.90**	**3.48**	14.54	2.34
	SAIL	0.70	3.18	**10.44**	**3.26**
Baking	DP-Fast	0.90	3.06	16.15	2.26
	SAIL	1.00	**4.20**	**14.39**	**2.54**
Folding Cloth	DP-Fast	0.10	-2.28	14.60	2.08
	SAIL	**0.30**	**-0.78**	**13.68**	**2.22**
Plate Fruits	DP-Fast	0.60	2.22	13.74	1.66
	SAIL	**0.80**	**5.46**	**8.53**	**2.67**
Pack Chicken	DP-Fast	0.40	0.51	17.33	1.25
	SAIL	**0.90**	**5.22**	**9.40**	**2.30**
Bimanual Serve	DP-Fast	0.40	1.00	12.01	1.43
	SAIL	**0.70**	**5.40**	**7.19**	**2.39**

A. Experiment Setup

Tasks. Our real-world experiment setup is shown in Figure 8. We choose 7 challenging tasks to demonstrate the effectiveness of our method. Task descriptions are provided in Appendix VII-D3.

Metrics. Similar to simulation experiments in subsubsection VII-G2, we evaluate our method on the real robot in terms of task success rate, average time over successful rollouts, speedup over demos, and throughput-with-regret, under different speedup conditions. TPR for the real evaluation is per-minute. Each method is evaluated for 10 rollouts per task.

Baselines. We compare SAIL with naive speedup of vanilla diffusion policy [7] termed **DP-Fast**. New observations are retrieved at the end of the sequence execution and used to perform the next inference step. When speeding up the robot, we change the command sending rate by adjusting the time interval δ as noted in section II.

B. Results and Discussion

Across differences in control systems, robot dynamics, and tasks, SAIL generally improves task throughput. As shown in Table II, throughput-with-regret (TPR) and SOD both

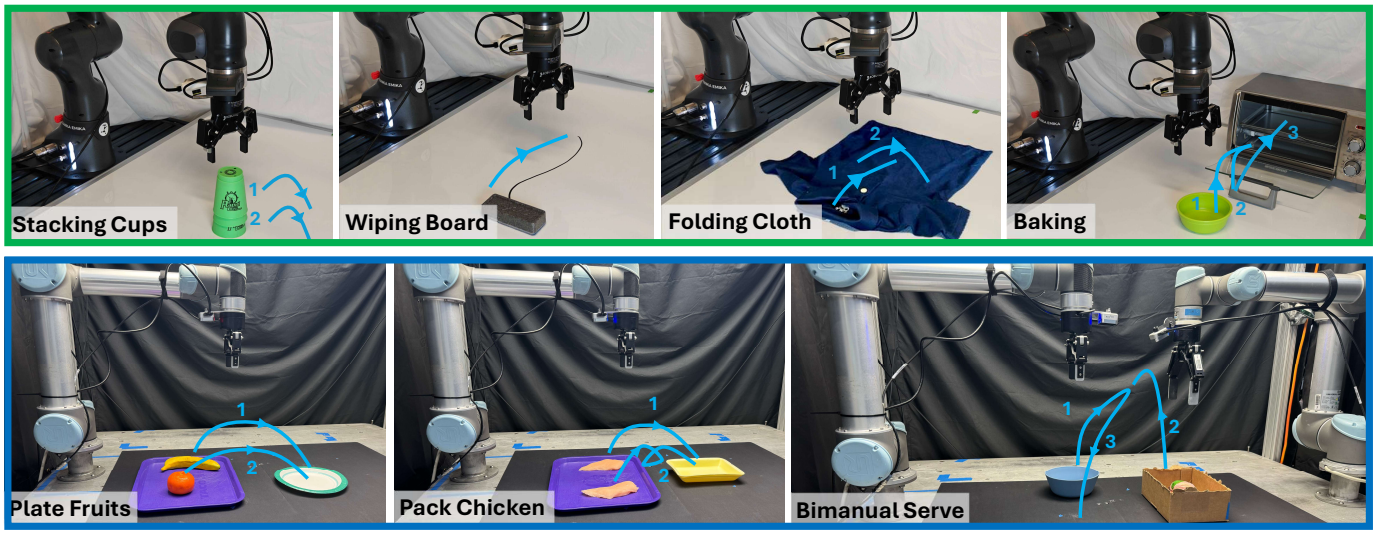


Fig. 8: Evaluation Task Suite Used in Real World.

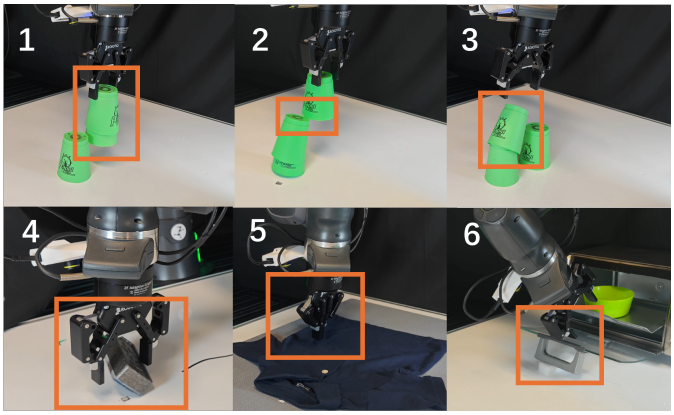


Fig. 9: **Commonly-seen failure modes from real-world evaluation.** Speeding up policy execution poses challenges unseen in normal speed execution, which include imprecise grasping (grasping two cups in 1, missing eraser in 4), low-fidelity tracking (colliding with other cups in 2 and 3, missing collar in 5, missing handle in 6). SAIL effectively lowers the frequency of failure under speeding-up execution, leading to a higher success rate and throughput compared to the baseline.

improved on 6/7 challenging tasks, demonstrating SAIL's consistent speed advantage over the baseline sped-up diffusion policy. Qualitatively (Figure 9), SAIL overcomes common DP failure modes during high-speed execution. DP often pauses due to **action depletion** when inference lags; SAIL maintains constant motion via smooth action scheduling. High speeds exacerbate **imprecise grasping** for DP, whereas SAIL's adaptive speed modulation slows critical phases, enhancing success rates in mid-to-high precision tasks such as plating fruits, packing chicken, and cup stacking. **Low-fidelity motion tracking** frequently causes DP failures, particularly for actions requiring precise localization like cloth folding and baking; SAIL substantially reduces tracking errors compared to the original teleoperation controller ($\mathcal{K}^{\text{teleop}}$), thus improving performance in precision-critical stages. Furthermore, DP's **inconsistent action predictions** result in jerky motions, posing considerable challenges at increased speeds. This is particularly evident in the bimanual serving task, where SAIL achieves smoother trajectories, yielding 5.4× higher throughput and approximately 1.8× greater success rates. Finally, we note that SAIL performs slightly worse than DP in the wiping task. We hypothesize that this is due to the high-gain tracking controller unable to adjusting to the new robot-object dynamics (sustained contact during wiping) at a higher speed.

VI. Conclusion

In this work, we addressed the critical challenge of enabling faster-than-demonstration execution for imitation learning policies. Our study reveals that achieving high-speed policy execution requires overcoming coupled challenges of tracking errors, distributional shifts, inconsistent motion prediction, and system latencies. The proposed framework, SAIL (Speed-Aaptive Imitation Learning), integrates high-gain feedforward control, consistent action prediction, and careful system design to ensure both precise tracking and consistent reference motion generation at high speeds. We complement this with adaptive speed modulation and action scheduling to handle real-world constraints. Our experiments demonstrate that SAIL achieves up to a 3.2× speedup across challenging manipulation tasks in real world and 4× speedup in simulation. We hope these insights into high-speed imitation learning provide a foundation for future research and real-world deployment.

Acknowledgments

The research presented in this paper was supported in part by the Agricultural Technology Research Program of the Georgia Tech Research Institute.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings*

of the twenty-first international conference on Machine learning, page 1, 2004.

[2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57 (5):469–483, 2009.

[3] S. Balasubramanian, A. Melendez-Calderon, and E. Burdet. A robust and sensitive metric for quantifying movement smoothness. *IEEE Transactions on Biomedical Engineering*, 59(8):2126–2136, 2012. doi: 10.1109/TBME.2011.2179545.

[4] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*, pages 783–792. PMLR, 2019.

[5] Daniel S Brown, Wonjoon Goo, and Scott Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations. In *Conference on robot learning*, pages 330–359. PMLR, 2020.

[6] Xuxin Cheng, Jialong Li, Shiqi Yang, Ge Yang, and Xiaolong Wang. Open-television: Teleoperation with immersive active visual feedback. *arXiv preprint arXiv:2407.01512*, 2024.

[7] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[8] 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. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. KDD'96, page 226–231. AAAI Press, 1996.

[10] 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.

[11] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[12] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[13] Balasubramanian, Sivakumar and Melendez-Calderon, Alejandro and Roby-Brami, Agnes and Burdet, Etienne. On the analysis of movement smoothness. *Journal of NeuroEngineering and Rehabilitation*, 12(1):112, 2015. doi: 10.1186/s12984-015-0090-9. URL https://doi.org/10.1186/s12984-015-0090-9.

[14] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50 (2):1–35, 2017.

[15] Aadhithya Iyer, Zhuoran Peng, Yinlong Dai, Irmak Guzey, Siddhant Haldar, Soumith Chintala, and Lerrel Pinto. Open teach: A versatile teleoperation system for robotic manipulation. *arXiv preprint arXiv:2403.07870*, 2024.

[16] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

[17] Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Maximilian Du, and Chelsea Finn. Bidirectional Decoding: Improving Action Chunking via Closed-Loop Resampling. *arXiv preprint arXiv:2408.17355*, 2024.

[18] Don O Loftsgaarden and Charles P Quesenberry. A nonparametric estimate of a multivariate density function. *The Annals of Mathematical Statistics*, 36(3):1049–1051, 1965.

[19] 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. In *5th Annual Conference on Robot Learning*, 2021.

[20] 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. In *7th Annual Conference on Robot Learning*, 2023.

[21] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[22] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[23] Harish Ravichandar, Athanasios S Polydoros, Sonia Chernova, and Aude Billard. Recent advances in robot learning from demonstration. *Annual review of control, robotics, and autonomous systems*, 3(1):297–330, 2020.

[24] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[25] Yuki Saigusa, Ayumu Sasagawa, Sho Sakaino, and Toshiaki Tsuji. Imitation learning for variable speed motion generation over multiple actions. In *IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–6. IEEE, 2021.

[26] Yuki Saigusa, Sho Sakaino, and Toshiaki Tsuji. Imitation learning for nonprehensile manipulation through self-supervised learning considering motion speed. *IEEE Access*, 10:68291–68306, 2022.

[27] Sho Sakaino, Kazuki Fujimoto, Yuki Saigusa, and Toshiaki Tsuji. Imitation learning for variable speed contact motion for operation up to control bandwidth. *IEEE Open Journal of the Industrial Electronics Society*, 3: 116–127, 2022.

[28] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[29] Lucy Xiaoyang Shi, Archit Sharma, Tony Z Zhao, and Chelsea Finn. Waypoint-Based Imitation Learning for Robotic Manipulation. In *Conference on Robot Learning*, pages 2195–2209. PMLR, 2023.

[30] Priya Sundaresan, Hengyuan Hu, Quan Vuong, Jeannette Bohg, and Dorsa Sadigh. What's the Move? Hybrid Imitation Learning via Salient Points. *arXiv preprint arXiv:2412.05426*, 2024.

[31] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[32] Philipp Wu, Yide Shentu, Zhongke Yi, Xingyu Lin, and Pieter Abbeel. GELLO: A General, Low-Cost, and Intuitive Teleoperation Framework for Robot Manipulators, 2023.

[33] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. Imitation learning from imperfect demonstration. In *International Conference on Machine Learning*, pages 6818–6827. PMLR, 2019.

[34] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 5628–5635. IEEE, 2018.

[35] 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.

[36] Yifeng Zhu, Abhishek Joshi, Peter Stone, and Yuke Zhu. VIOLA: Imitation Learning for Vision-Based Manipulation with Object Proposal Priors. *arXiv preprint arXiv:2210.11339*, 2022. doi: 10.48550/arXiv.2210.11339.

[37] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A Modular Simulation Framework and Benchmark for Robot Learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[38] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

VII. APPENDIX

A. Related Work

In this section, we first briefly summarize offline imitation learning. We then discuss the state of the art and challenges in improving imitation learning beyond demonstrations in terms of both task performance and robot speed. Finally, we note challenges in system integration for imitation learning.

*1) **Offline Imitation Learning**:* Imitation learning, or learning from demonstration (LfD), is a common way of programming robots using only demonstrations of the desired robot behavior [22, 2, 14, 23]. One specific paradigm of imitation learning is behavior cloning (BC) [23], where a function approximator, typically a deep neural network, is trained on demonstrations to map observations to actions [34]. Recent works have adopted deep generative models, particularly Diffusion Models, to better capture the rich behavior distributions present in real-world demonstration data. However, as noted by prior works [35, 17], sampling from these learned trajectory distributions can break temporal dependencies between consecutive prediction steps. This challenge is further exacerbated in our setting, where higher execution speeds create additional distributional shifts. To address this, we develop a consistency-preserving trajectory generation component that explicitly maintains temporal coherence during high-speed execution.

Furthermore, an implicit assumption made by most offline imitation learning methods, regardless of the architecture design, is that the overall robot system behaves identically during both data collection and policy execution. However, as we demonstrate empirically, this assumption breaks down when increasing execution speed due to changed dynamic responses and controller behaviors. Our method addresses this limitation by training policies to predict the actual end-effector poses achieved by the robot during demonstrations, rather than the commanded poses sent to the controller. This approach, combined with high-fidelity tracking control, enables the policy to better account for the robot's dynamic response and maintain consistent behavior across different execution speeds.

*2) **Better-than-Demonstration Imitation Learning**:* Several imitation learning approaches aim to perform tasks better than the expert demonstrations [4, 5, 33]. For instance, T-REX [4] learns better-than-demonstration policies by using a ranking-based reward function to evaluate unseen policy behaviors. However, these methods typically operate in an Inverse Reinforcement Learning [21, 1, 38] setting, requiring interactive learning in the environment.

In contrast, our focus is on the purely offline setting, where we execute an offline-learned policy faster during runtime. While prior works generally emphasize improving task success rates, we aim to maximize *task throughput*, defined as the number of successful executions per unit of time and perform tasks faster than the teacher did during demonstrations.

Recent works, such as SPHINX [30], have shown modest improvements in execution time and speedup. However, these gains are typically byproducts of their methods, whereas we explicitly target faster execution as a primary objective.

*3) **System Integration for Imitation Learning**:* Recent works have expanded beyond pure algorithmic innovations to develop full-stack systems for imitation learning [35, 10, 6, 15, 8]. Most existing research focuses on two directions of innovation: building new teleoperation devices for more effective data collection, and designing new learning algorithms to better leverage such data. Systems such as Mobile-ALOHA [10], Universal Manipulation Interface (UMI) [8], and OpenTelevision [6] have shown the tremendous value in full-stack design of imitation learning system that integrates low-level robot controllers with learning algorithms. Most relevant to us is UMI [8], which explicitly tackles the challenges of system latencies, originating from various sources such as sensing and inference delays, in order to minimize out-of-distribution input to the policy. However, none of these works consider the problem of deliberately varying the execution speed between demonstration and policy execution. The core contribution of our work lies in identifying key challenges arising from this new problem setting and a system that enables faster-than-demonstration policy execution.

B. Evaluation Metrics

In this section, we describe the metrics that we used for evaluation in detail.

1) **SR** (higher is better): The *success rate* is the number of successfully-completed tasks divided by the total number of trials.
2) **TPR** (higher is better): We propose *throughput-with-regret* to reward faster successes while penalizing all failures equally:

$$\mathrm{TPR} = \frac{1}{N}\sum_{i=1}^{N}\left(\left(\frac{1}{t_i}\cdot S_i\right) - \left(\frac{1}{t^{\max}}\cdot(1-S_i)\right),\right) \tag{6}$$

where t_i is the *duration* (i.e., total simulated clock time) of rollout i, $t^{\max}$ is the maximum amount of time allowed per trial, and S_i is the *success* of rollout i. That is, $S_i = 1$ if trial i was successful and 0 otherwise. We halt all trials if they have not succeeded before $t^{\max}$, and declare such trials as failures.
3) **ATR** (lower is better): We record the *average time for successful rollouts*, where we compute the average time in seconds only for the successful rollouts out of all trials.
4) **SOD** (higher is better): We report the *speedup-over-demo*, which is the average length of a demo divided by ATR. SOD indicates how much one speeds up the execution of the imitation learning policy compared to the training demonstration.
5) **CON** (lower is better): To evaluate our CFG and action conditioning approach, we quantify the *consistency* between overlapping parts of consecutive action sequences, we measure the change in actions at the transition point, specifically, $\mathrm{CON} = \hat{a}_{H^{\mathrm{f}}} - a_{H^{\mathrm{e}}+H^{\mathrm{f}}}$, following the notation of Section III-B.
6) **SPARC** (higher is better): SPARC (linear and angular spectral arc length) is a smoothness metric that evaluates the arc length of the Fourier magnitude spectrum of a

trajectory's speed profile [13]. It is an extended version of Spectral Arc Length (SAL) [3].

In SAL, the magnitude spectrum $V(\omega)$ of the Fourier transform of a speed profile v_t is normalized by its DC value $V(0)$:

$$\hat{V}(\omega) = \frac{V(\omega)}{V(0)} \tag{7}$$

SAL integrates the arc length of $\hat{V}(\omega)$ over frequencies from 0 up to a cutoff frequency ω_c:

$$\text{SAL} \triangleq -\int_0^{\omega_c} \left[\left(\frac{1}{\omega_c} \right)^2 + \left(\frac{d\hat{V}(\omega)}{d\omega} \right)^2 \right]^{\frac{1}{2}} d\omega \tag{8}$$

where the first term in the square root is used for frequency normalization, normalizing the arc length with respect to ω_c. SPARC refines SAL by adaptively selecting ω_c based on a chosen amplitude threshold $\overline{V}$ and an upper limit $\omega_c^{\max}$ as follows:

$$\omega_c \triangleq \min\Big\{ \omega_c^{\max}, \min\big\{ \omega, \hat{V}(r) < \overline{V} \ \ \forall r > \omega \big\} \Big\}. \tag{9}$$

We compute SPARC for a given speed trajectory in the following manner. First, we pad the trajectory with zeros ($K = 4$) to increase the frequency resolution to accurately estimate the length of the arc. Next, we normalize the magnitude spectrum and apply an upper limit $\omega_c^{\max} = 20$ and an amplitude threshold $\overline{V} = 0.05$. We then compute the arc length of the normalized spectrum by summing the Euclidean distance between successive frequency-domain points. Finally, we multiply this sum by -1 to obtain larger values for smoother trajectories.

7) **LDLJ** (lower is better): Log Dimensionless Jerk (LDLJ) is a smoothness metric that evaluates how quickly and drastically the motion accelerates or decelerates based on the third derivative of position, jerk.

   LDLJ can be written as [13]:

$$\text{LDLJ} \triangleq -\ln \left| -\frac{(t_2 - t_1)^5}{(v^{\text{peak}})^2} \int_{t_1}^{t_2} \left| \frac{d^2 v_t}{dt^2} \right|^2 dt \right| \tag{10}$$

   where t_1 and t_2 are the start and end times of the movement, v_t is the speed at time t, and $v^{\text{peak}} \triangleq \max_{t \in [t_1, t_2]} v_t$. In our work, we calculate LDLJ by setting v^{peak} as the peak speed within the trajectory. The speed $v(t)$ is computed as the difference between the positions of successive points. After estimating the speed, we apply finite differences to approximate its second derivative with respect to time. The squared second derivative is then integrated over the movement duration, scaled by $\frac{(t_2 - t_1)^5}{v_{\text{peak}}^2}$, and the negative natural logarithm is applied to obtain the final LDLJ value.

8) **WED** (lower is better): We calculate the *Weighted Euclidean Distance*, a metric that is used in [17] to quantify the consistency between overlapping action segments.

C. Simulation Experiment Detail

In this section, we detail the controller setup and data collection pipeline in for simulation experiment.

*1) **Robot Control and Dynamics Considerations**:* We control robots in simulation using an OSC controller that takes absolute pose commands, except for some of our baselines, which use delta pose commands. Additionally, to ensure that robot torque limits are not a bottleneck to speed up, we removed the joint torque limits of the Franka Emika Panda robot in Robosuite. For some of our baselines, removing torque limits resulted in worse performance. In these cases, we report their performance with torque limits. In Table VI, we list the optimal controller and the upper bound of c in adaptive speed modulation that SAIL uses in the simulated tasks.

*2) **Simulator and Data**:* We use Robosuite [37] to simulate robots and their environments. Robosuite is built on Mujoco [31] and by default simulates two milliseconds (0.002 s) of real-world time every time the simulator is stepped forward. In our problem setting, we consider the action interval δ as the number of simulation steps allowed for a robot controller to execute a given action. Speeding up policy execution is thus to reduce the total number of simulation step taken to finish a task. The teleoperation data is collected at $\delta = 0.05$ s (20 Hz).

For our simulation benchmark, we use three tasks from the Robomimic [19] suite of tasks and two tasks from MimicGen [20]. For the Robomimic tasks, we train a separate policy for each task on 200 human demonstrations. For MimicGen tasks, we use 500 machine-generated demonstrations for each task.

The diffusion policy is trained with prediction horizon $H = 32$. In a receding-horizon manner, we execute 8 of the predicted actions before the next inference, and 4 more actions while inference is running to simulate sensing-inference delay.

*3) **Compute**:* We run all sim experiments on a compute cluster. Each experiment uses a single A40 GPU, 8 CPU cores and 64GB of RAM.

D. Real-World Evaluation Setup

In this section, we explain the real robot setup and data collection pipeline used in the paper.

*1) **Franka Robot**:* We have a four-level control hierarchy for the Franka robot. In the first level, action chunks from policy inference are retrieved in a variable frequency and velocity approximation is performed, as introduced in III-D. Second, actions in each chunk are interpolated and scheduled by a computer (Intel NUC) controlling the robot at 100Hz. Third, we use our OSC controller for the Franka, which is based on the Deoxys controller introduced in [36]. Fourth and finally, we leverage the torque control API from libfranka and calculate torque commands, which are sent to the on-board Franka controller at 500Hz.

*2) **Data Collection**:* Our teleoperation system uses a Meta Quest VR headset to control the *commanded pose* of the robot end effector. To record an initial pose during the demonstration collection, users press the VR controller grip button. While

the button is held, the change in the VR controller pose relative to the initial pose is transformed into the robot coordinate frame and used to adjust the *commanded pose*. Releasing the grip button pauses the teleoperation, allowing users to reposition their hand comfortably before resuming the task. We collect the robot's *reached poses* and *commanded poses* at 100Hz. Images are attained from the Kinect camera which is collected at 30Hz, while Zed camera is collected at 60Hz. Demonstrations are recorded at 20Hz. During each data acquisition, we use the latest timestamp for the third-person view camera as the observation timestamp. We then retrieve the latest wrist camera observation and interpolate proprioceptive observation according to the observation timestamp. We collect 50 demonstrations for each task.

*3) **Task Descriptions**:*

- **Stacking Cups**. This task mimics speed stacking, wherein humans attempt to stack cups in predetermined sequences as quickly as possible. We collect human teleoperation demos to stack 3 cups into a pyramid shape for this task. The repetitive grasping, placing, and movement pose challenges in balancing speeding up policy and manipulation accuracy.
- **Baking**. The baking task represents use cases in a commercial kitchen where efficiency are important. The goal for this task is to pick up a bowl from a table and place it precisely on an oven rack. Then the robot must close the oven door, which requires precise contact-rich motion to accomplish.
- **Folding Cloth**. This task requires the robot to fold a t-shirt. The robot must pick the collar and fold the whole shirt in half, then pick the right sleeve to do another fold. Success requires finishing both folds.
- **Wiping Board**. To showcase SAIL's robustness we include a contact-rich manipulation scenario: the wiping-board task. The robot must pick up an eraser and wipe a line on a whiteboard, all while maintaining forceful contact. Since the demonstration data does not include any force and contact information, accelerating execution of such a task with imitation learning is challenging.
- **Plate Fruits**. This task consists of picking two fruits from a random position on a tray and placing them on a plate with a specific pattern. This task is challenging because the picking order is fixed, but the positions can be swapped.
- **Pack Chicken**. This real world packing task is challenging because of the variability in the shape of the deformable (rubber) chicken breasts and the limited size of the container requiring precise placements. The robot is required to rotate the second chicken breast to fill the space in the container. This complex motion makes speeding up challenging.
- **Bimanual Serve**. This task involves two robots operating together. While the first is picking a peach, the second is picking a bowl. Then, they converge to a common point where the first robot places the peach in the bowl. Then the bowl with the peach inside is served. This task is very difficult to accelerate because it requires alignment and synchronization between the two arms.

E. Deviation of Lower Bound δ^{lb} for Action Interval

We seek to derive a lower bound for the action interval δ (i.e., highest speed up) that still allows a continuous control loop. Recall that H^{p} is the prediction horizon of the policy, and let δ^{lb} be the (constant) lower bound on δ_t that we aim to find. To find it, we consider the three critical parameters: (a) Sensing-inference delay $\delta^{\text{delay}} = t^a - t^o$, (b) the shortest-possible action execution time $H^{\text{p}} \cdot \delta^{\text{lb}}$, and (c) the shortest-possible length of the action chunk used for consistency-preserving conditioning H^{c} (subsection III-B). To ensure continuous execution, we require the available action sequence to be longer than the sensing-inference delay plus the conditioning horizon, which gives the lower bound δ^{lb} as

$$\begin{aligned} & H^{\text{p}} \cdot \delta^{\text{lb}} > \delta^{\text{delay}} + H^{\text{c}} \delta^{\text{lb}} \\ \implies \quad & \delta^{\text{lb}} > \frac{\delta^{\text{delay}}}{H^{\text{p}} - H^{\text{c}}}. \end{aligned} \tag{11}$$

We illustrate this relationship in Figure 6. Note that we can reduce δ^{lb} (i.e., allow higher speedup) by extending the prediction horizon H^{p}, but this would require accurate action prediction over a longer horizon, which is inherently challenging [16]. Hence, in practice, the prediction horizon H^{p} and the sensing-inference delay δ^{delay} jointly determines the minimum bound on δ_t and in turn the speedup factor c_t:

$$c_t \in \left(\delta^{\text{lb}} / \delta^*, 1 \right]. \tag{12}$$

Moreover, note that c_t from subsection II-D does not affect this computation, since the lower bound provides a worst-case guarantee—even though c_t may increase to slow down execution in certain phases, we cannot rely on this a priori when computing δ^{lb} for continuous execution.

F. Adpative Speed Modulation

Identifying Critical Actions via Motion Complexity. Inspired by Automatic Waypoint Extraction (AWE) [29], we approximate the commanded robot end effector poses in a demonstration with a set of waypoints connected by linear segments. With a set error budget, this algorithm produces more waypoints for more complex motion. We then identify fast and slow regions by clustering the waypoints in 3-D space using DBSCAN [9], which is well-known to identify arbitrarily-sized clusters better than spherical or centroid-based clustering methods like k-means; it also implicitly filters out noisy data points that may not represent significant motion changes. To correlate each time step t with a waypoint, we linearly interpolate between the waypoints in time. Finally, given a minimum cluster size, we label each time-interpolated waypoint with $k_t = 1$ if the waypoint at time t is in a cluster and $k_t = 0$ otherwise.

Identifying Critical Actions via Gripper Events. We observe that motions involving critical actions often occur during interactions with objects and the environment, which are correlated with gripper state changes. Thus, we use these

gripper events to identify the critical actions. That is, we set $k_t = 1$ if the gripper is changing (opening or closing) at t and $k_t = 0$ otherwise.

G. Experiments for Testing Hypothesis 1-3

*1) Testing **H1**: Speeding Up Policy Execution Requires a High-Gain Controller:* To illustrate how controller tracking performance affects task execution during runtime, we replay the demonstrations of the Can task at different speeds and record the success rate for a given controller gain (K_p). We compare replaying the commanded poses (x^{d}) and reached poses (x) in demonstrations. As shown in Figure 10, **High-gain control combined with reached pose tracking enables consistent behavior across execution speeds.**. Commanded poses lead to overshooting and task failures when attempting faster execution with higher gains. In contrast, using reached poses as reference trajectories enables consistently high success rates at increased speeds, provided the controller gain is sufficiently high to ensure accurate tracking.

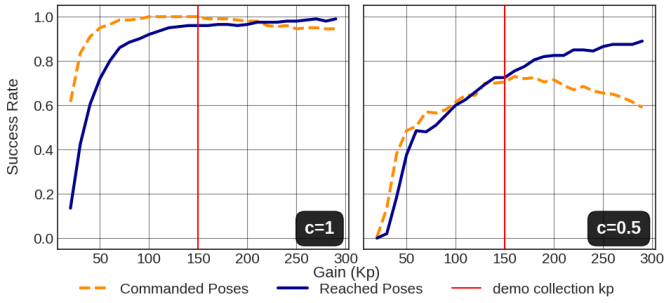


Fig. 10: **Demo replay at different speeds and controller gains.** We examine the effects of increasing controller gains and speed for replaying demos in simulation. Left: using commanded poses performs better when replaying at the original speed ($c = 1$) but using reached poses matches performance when using high gains. Right: A high-gain controller using reached poses performs better than one using commanded poses at a higher execution speed.

*2) Testing **H2**: A High-Gain Controller Requires a Smooth Reference Trajectory:* In this experiment, we assess task success rate versus increasing noise scale. This is important because faster execution can result in out-of-distribution observations that cause a policy to produce noisier reference trajectories. **High-gain controllers are more sensitive to noisy reference trajectories.** As seen in Figure 11, as the noise scale increases, the high-gain controller's performance deteriorates more rapidly than the low-gain controller. This reveals an important coupling in our system: while high-gain control is necessary for accurate tracking at increased speeds, it also amplifies any inconsistencies in the predicted reference trajectories. This explains the need for trajectory smoothing in our proposed method.

*3) Testing **H3**: Action Conditioning Improves the Temporal Consistency of Across Predictions:* Since inconsistent action predictions are the main reason for unsmooth reference trajectories and failure in faster execution, we need to test whether action conditioning can improve temporal consistency and smoothness of generated actions. To quantify this improvement, we conduct rollouts with and without action conditioning and evaluate smoothness and consistency using the SPARC, CON, and BID metrics (see Appendix VII-B for details). We assess these metrics across five tasks at a speed-up factor of 0.2. As delineated in Table III, we found that the action conditioning results in smoother actions, supported by the higher SPARC metric and lower CON, BID metric compared to baseline DP.

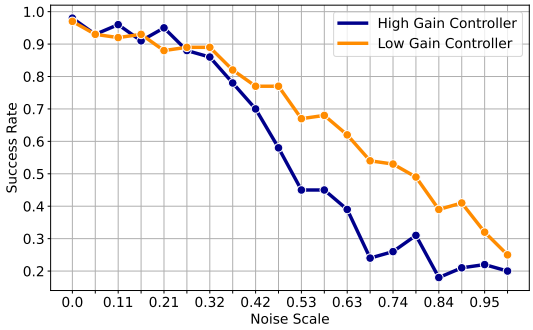


Fig. 11: **Noise vs high-gain and low-gain controller.** We study the effects of increasingly noisy actions with a high-gain and low-gain controller. Success Rate is averaged over 100 rollouts per noise scale. At higher noise levels, the success rate when using a high-gain controller drops more significantly than with a low-gain controller.

TABLE III: **Evaluation of smoothness of actions**. We compare the smoothness of the generated actions using our method (c=0.2)

		SR ↑	SPARC ↑	CON ↓	WED ↓
Lift	SAIL	**0.97**	**-2.80**	**0.091**	**0.348**
	BID[17]	0.53	-2.85	0.116	0.395
	Baseline	0.92	-2.83	0.094	0.376
Can	SAIL	**0.76**	**-2.80**	0.232	0.885
	BID[17]	0.62	-2.93	0.325	**0.525**
	Baseline	0.73	-2.83	**0.230**	0.930
Square	SAIL	**0.87**	**-2.74**	**0.107**	0.916
	BID[17]	0.12	-3.06	0.217	**0.327**
	Baseline	0.81	-2.80	0.121	0.957
Stack	SAIL	0.90	**-2.50**	0.168	**0.634**
	BID[17]	**0.92**	-2.56	0.641	0.794
	Baseline	0.89	-2.56	**0.156**	0.739
Mug	SAIL	**0.63**	-2.80	**0.228**	1.192
	BID[17]	0.40	**-2.62**	0.831	**0.679**
	Baseline	0.59	-2.88	0.276	1.210

H. CFG Experiments

In additional experiments, we validate the key insights that motivated the design of consistency-preserving action prediction generation. Specifically, we test the following hypotheses:

- ***H-CFG1***: Consistency-guiding is beneficial when action condition is aligned with observation
- ***H-CFG2***: Policy speed-up results in more misalignment between observation and action condition

- ***H-CFG3***: Tracking error is highly correlated to observation-action misalignment

*I. Testing **H-CFG1**: Consistency-guiding is beneficial when action condition is aligned with observation*

We provide an additional study on the key factors influencing our Classifier-Free Guidance (CFG) approach for preserving action consistency. Namely, we test the hypothesis ***H-CFG***: consistency guiding is effective when conditioned on future action condition in unconditional action distribution.

In this experiment, we investigate when consistency guidance is beneficial, specifically when the future action condition belongs to the unconditional action distribution. We evaluate how well the generated actions conform to the future action condition under different perturbations.

Our key argument is that consistency guidance is most effective when the future action condition exists within the unconditional action distribution. To validate this, we conduct the following experiment. Given a fixed observation, we sample 64 action sequences from the unconditional action distribution by setting the future action condition to a null token. Next, we select one of these samples as the future action condition and generate 64 action sequences using Classifier-Free Guidance (CFG) with a weight of 1. The resulting conditional action distribution is shown in the left column of Figure 12.

To analyze the effects of perturbations, we apply two modifications. First, we introduce a temporal shift by delaying the actions, as shown in the center column. Second, we introduce spatial perturbations by adding uniformly sampled noise (range: 0.02) to the selected actions, as shown in the right column.

Our results indicate that CFG performs best when the future action condition is within the unconditional action distribution. This suggests that such conditions are likely present in the training dataset, meaning the model has encountered similar action-observation pairs during training. Consequently, the model learns to correctly condition on these actions. However, when the future action condition deviates from the unconditional distribution—potentially due to tracking errors—the model may struggle to compute an appropriate conditional score.

J. Aggregating actions

We describe the algorithm for aggregating actions, which is one of our baselines, in Algorithm 1.

Algorithm 1: Aggregate Actions

Input: Sequence of Delta Cartesian actions A
Output: Sequence of Aggregated Actions *AggActions*
$AggActions \leftarrow []$
$CurrAction \leftarrow A[0]$
for each element a in $A[1:]$ **do**
  if magnitude($CurrAction$) $>$ 0.05cm **then**
    Append *CurrAction* to *AggActions*
    $CurrAction \leftarrow a$
  else if DotProduct(a, $CurrAction$) $<$ 0.25 **then**
    Append *CurrAction* to *AggActions*
    $CurrAction \leftarrow a$
  else
    $CurrAction \leftarrow CurrAction + a$
  end if
end for
Append *CurrAction* to *AggActions*
Return *AggActions*

K. Controller Design

We use Operation Space Controller for both simulation and physical hardware. Given the desired 6D pose, and velocity, we calculate the pose error e_p and velocity error e_v. The computed torque τ sent to robot joints is

$$\tau = J^\top M(K_p e_p + K_v e_v), \tag{13}$$

where J is the Jacobian of the robot, M is the mass matrix represented in end-effector space. The velocity target is computed by fitting and differentiated a spline over the predicted action trajectory. We use the velocity FF term only in the real world, as position-only tracking is sufficient for simulated environment.

*L. Testing **H-CFG2**: Policy speed-up results in more misalignment between observation and action condition*

We now examine how policy speed-up influences action conditioning. Specifically, we hypothesize that increasing policy speed-up results in the action condition deviating further from the unconditional action distribution, potentially degrading the performance of CFG.

To validate this, we conduct the following experiment. We construct a batch of scenarios consisting of the simulation's internal state and corresponding observations from expert demonstrations. For each scenario, we reset the simulation to the recorded internal state and use the diffusion policy to generate actions $a_{0:H}$. Following the notation in Section III-B, we execute $a_{0:H^{\mathrm{e}}}$ using different policy speed-up factors, and obtain the resulting observation $o_{H^{\mathrm{e}}+1}$. Notably, the pair $(o_{H^{\mathrm{e}}+1}, a^{\mathrm{c}} = a_{H^{\mathrm{e}}:H^{\mathrm{e}}+H^{\mathrm{f}}})$ represents the input to SAIL for CFG-based action generation. Instead of performing generation, we assess how well the action condition aligns with the observation.

To quantify this alignment, we follow the methodology described in Subsection VII-I, where we sample $N = 64$ action sequences from the unconditional action distribution. We then compute in-distribution scores using standard out-of-distribution (OOD) techniques:

1) **Kernel Density Estimation (KDE)** [28]: We estimate the likelihood of the action condition under the empirical distribution using a Gaussian kernel, with the bandwidth adaptively selected via Scott's rule. Higher values suggest better in-distribution.
2) **k-Nearest Neighbors (kNN) Distance** [18]: We quantify how close the action condition is to its nearest neighbors

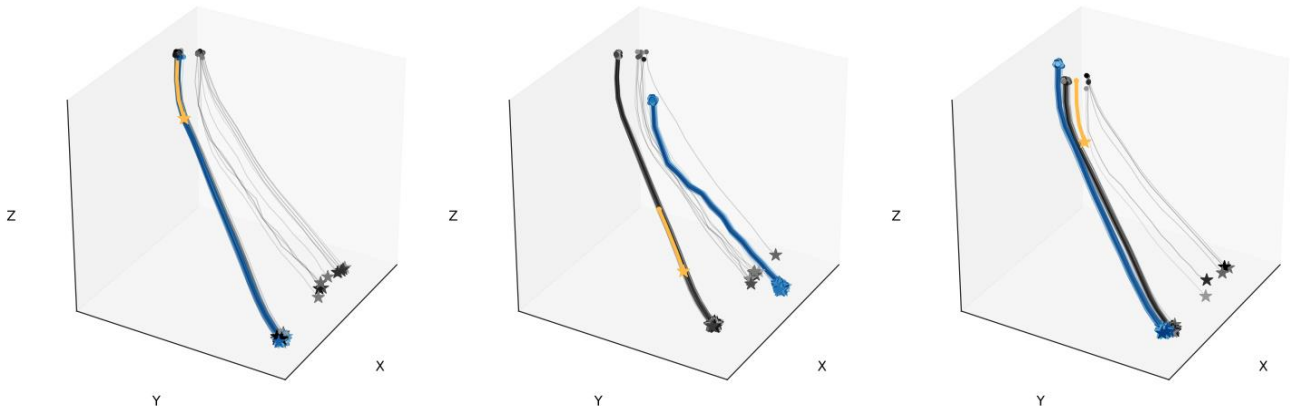


Fig. 12: **Impact of Action Conditioning on Consistency Guidance.** We evaluate how our algorithm adheres to conditioning on predicted actions. The action conditioning is shown in yellow; samples from the unconditional action distribution are shown in grey; and the conditional action distribution is shown in blue. The left panel shows future action conditions sampled from the unconditional distribution. The center panel shows temporally shifted future action conditions, while the right panel shows spatially perturbed future action conditions. When the action conditioning (yellow) lies within the unconditional action distribution (grey) in both space and time, the conditional action distribution (blue) is consistent with the action conditioning. However, when the action conditioning falls outside the unconditional action distribution, the model struggles to align the generated actions with the given condition. The middle subplot shows time-shifted action conditioning and the right subplot shows space-shifted action conditioning.

in the dataset. Specifically, it is computed as the average Euclidean distance to the $k = 8$ nearest samples. Lower values suggest better in-distribution.

3) **Maximum Mean Discrepancy (MMD)** [11]: We compute the discrepancy between the unconditional action distribution and the action-condition (modeled as a Dirac delta) using a Gaussian kernel with a bandwidth of 0.5. Lower values suggest better in-distribution.

A key aspect of this experiment is that we reset the simulation before each rollout and evaluate only a single receding horizon step. This design isolates the direct effect of policy speed-up on action conditioning, avoiding confounding influences such as accumulated errors from prior rollouts. If the analysis were performed over an entire task execution, it would be difficult to disentangle whether action condition mismatches stem from speed-up itself or from historical execution deviations.

We compute these metrics across 200 scenarios and visualize the density estimates in Figure 13. Our results confirm that as the policy speed-up factor increases, action conditions are more likely to fall outside the unconditional action distribution. This trend suggests that at higher speeds, previously executed actions become less representative of the policy's expected next steps, leading to inconsistencies in action conditioning, and possible degradation of CFG performance. Hence, we would require the need for adaptive mechanisms to mitigate conditioning mismatches in high-speed policy execution.

*M. Testing **H-CFG3**: Tracking error is highly correlated to observation-action misalignment*

Previous results indicate that action conditioning is most effective when the action condition is well-aligned with the current observation. However, as policy speed-up increases, this alignment deteriorates, reducing the benefits of conditioning. To effectively determine when action conditioning is beneficial, we need a reliable and efficient proxy for measuring misalignment.

A natural approach is to use the out-of-distribution (OOD) metrics introduced in VII-L. However, these methods are computationally expensive and impractical for real-time deployment. Instead, we propose tracking error as a computationally efficient alternative.

Defining Tracking Error. Given the current robot state x and the desired state indicated by the action a, we define the position tracking error and orientation tracking error as follows:

$$e_{pos} = \left\| a_{pos} - x_{pos} \right\|$$
$$e_{ori} = cos^{-1}\left(\frac{tr(R_{\triangle}) - 1}{2}\right), \quad R_{\triangle} = R(a_{ori})^{\top} R(x_{ori})$$

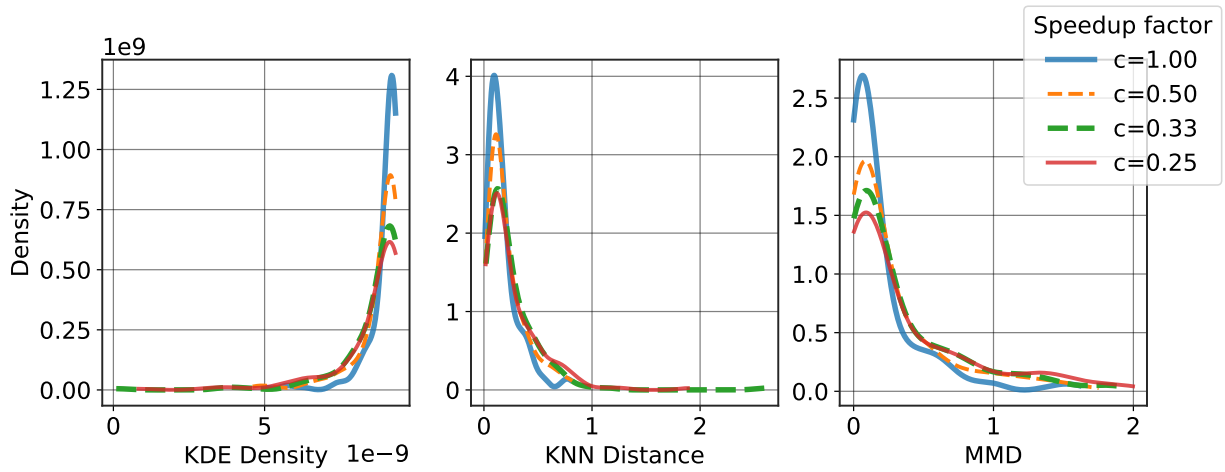


Fig. 13: **Effect of Policy Speed-up on Out-of-Distribution Action Conditions.** This figure evaluates how increasing the policy speed-up factor leads to mismatches between the action condition and the unconditional action distribution. We assess this by computing how well action conditions derived from previously executed actions align with the unconditional action distribution given the current observation. We use three independent OOD detection metrics: Kernel Density Estimation (Left), k-Nearest Neighbor Distance (Middle), and Maximum Mean Discrepancy (Right). Across different metrics, we observe that as the policy speed-up factor increases, the action condition increasingly falls into the OOD region, exhibiting a long-tail distribution in high-OOD regions. Comparing normal-speed policies (blue) and highly accelerated policies (red) reveals that slower policies maintain better alignment between action conditions and current observations.

where x_{pos} and a_{pos} denote the real and desired end-effector positions, x_{ori} and a_{ori} represents the real and desired end-effector orientations. $R(\cdot)$ maps any oreintation representation to an SO(3) rotation matrix, and $\mathrm{tr}(\cdot)$ denotes the trace of a matrix.

Analyzing Correlation with OOD Scores. To assess whether tracking error serves as a reliable indicator of action condition misalignment, we compare tracking error values against the action condition's in-distribution scores computed using the metrics defined in Section VII-L. Specifically, we analyze the correlation separately for position and orientation tracking errors.

Figure 14 visualize these relationships. The results indicate a clear trend: as tracking error increases, the action condition is more likely to fall outside the unconditional action distribution. This suggests that tracking error is a useful proxy for detecting when action conditioning might degrade CFG performance.

Impact of Adaptive CFG on Policy Performance. We evaluate whether adaptively applying CFG based on tracking error improves policy performance, particularly success rate and trajectory smoothness. To study the effect of different tracking error thresholds, we vary the position tracking error threshold across 0.01, 0.02, and 0.04 and evaluate the Square task with a policy speed-up factor of $c = 0.33$. Each policy is tested over 50 scenarios, with three evaluations per scenario.

CFG is applied only when the tracking error is below the specified threshold, and we compare this approach to a baseline without guidance. Performance is measured by success rate and smoothness using the SPARC metric. The proportion of inference steps where CFG was applied—referred to as the guided inference ratio—was 0.27, 0.47, and 0.78 for the three thresholds, respectively.

The results show that indiscriminate application of CFG (high guidance ratio) reduces success rate, likely due to misalignment between observation and action condition. Conversely, selectively applying CFG when the tracking error is low improves both success rate and smoothness. Figure 15 illustrates that a moderate tracking error threshold leads to better overall performance, while higher thresholds degrade success rates.

One limitation is that the optimal tracking error threshold varies by task, as tasks with more complex reference trajectories naturally exhibit higher tracking errors. The values reported here were determined through hyperparameter tuning for the Square task. Nonetheless, these findings confirm that tracking error provides a useful heuristic for determining when to apply CFG, leading to more reliable policy execution.

N. Hyperparameters

The hyperparameters for our Policy backbone are listed in Table IV. The parameters for the consistency guiding is listed in Table V and the controller parameters in simulation are listed in Table VI. The controller parameters for the real robot are listed in Table VII.

O. Detailed Ablation:

A more detailed ablation of our method is provided in Table VIII.

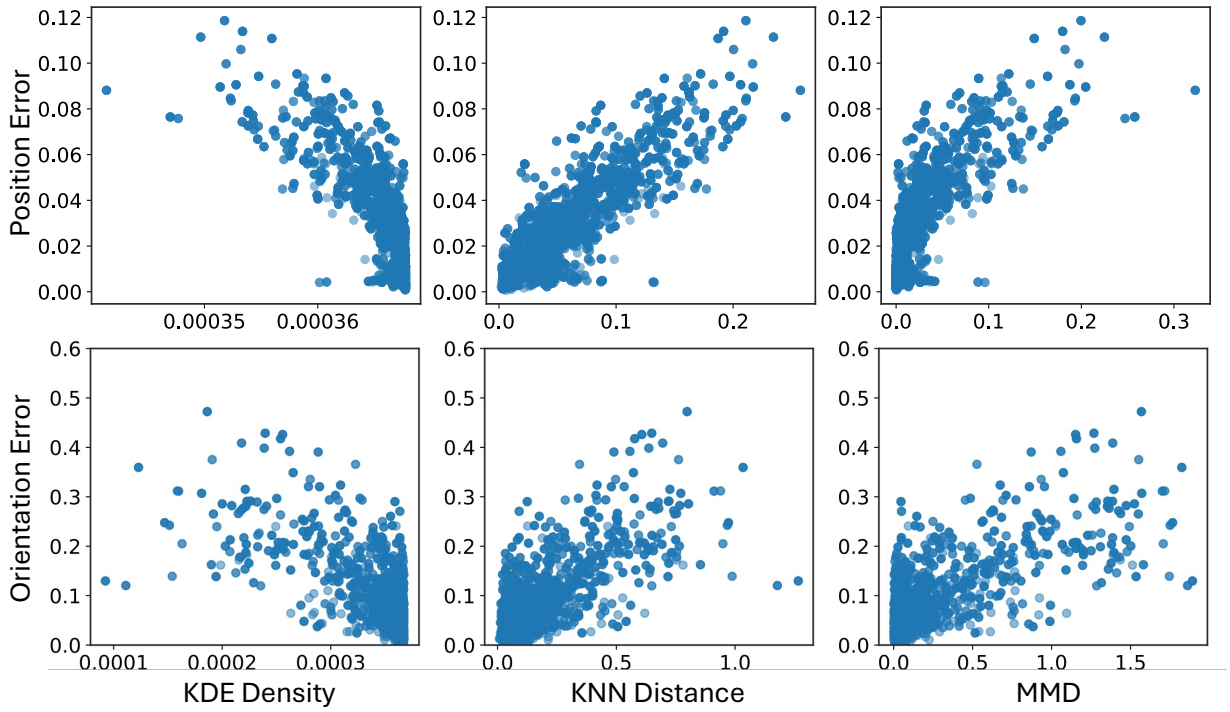


Fig. 14: **Correlation between Tracking Error and Out-of-Distribution Action Condition.** This figure illustrates the relationship between the tracking error and out-of-distribution (OOD) scores of action-conditions, computed using KDE Density (left), KNN distance (center), and MMD (right). The top row shows the correlation between the position tracking error and OOD scores in the position dimension. The bottom row shows the correlation between orientation tracking error and OOD scores in the orientation dimension. The result indicates that large tracking error corresponds to higher OOD scores, suggesting that tracking error can serve as a proxy for detecting misaligned action conditions.

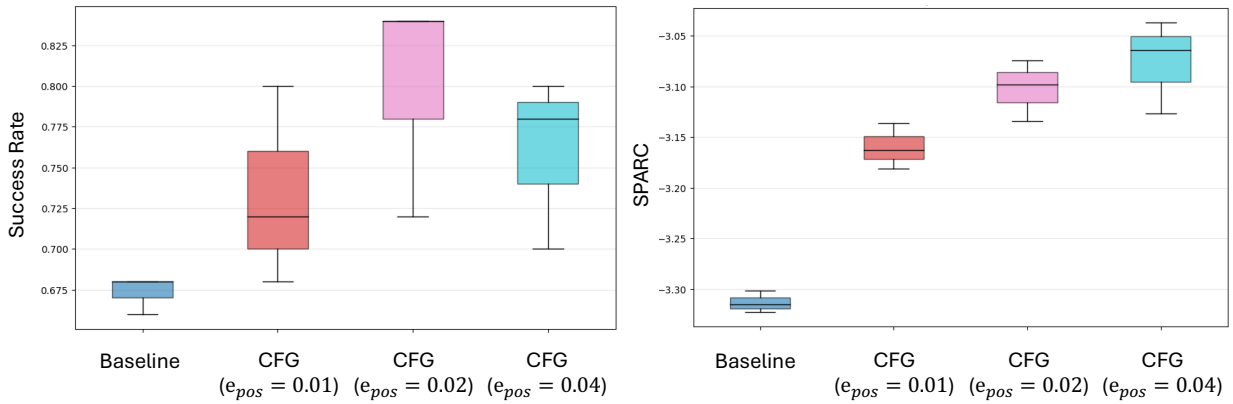
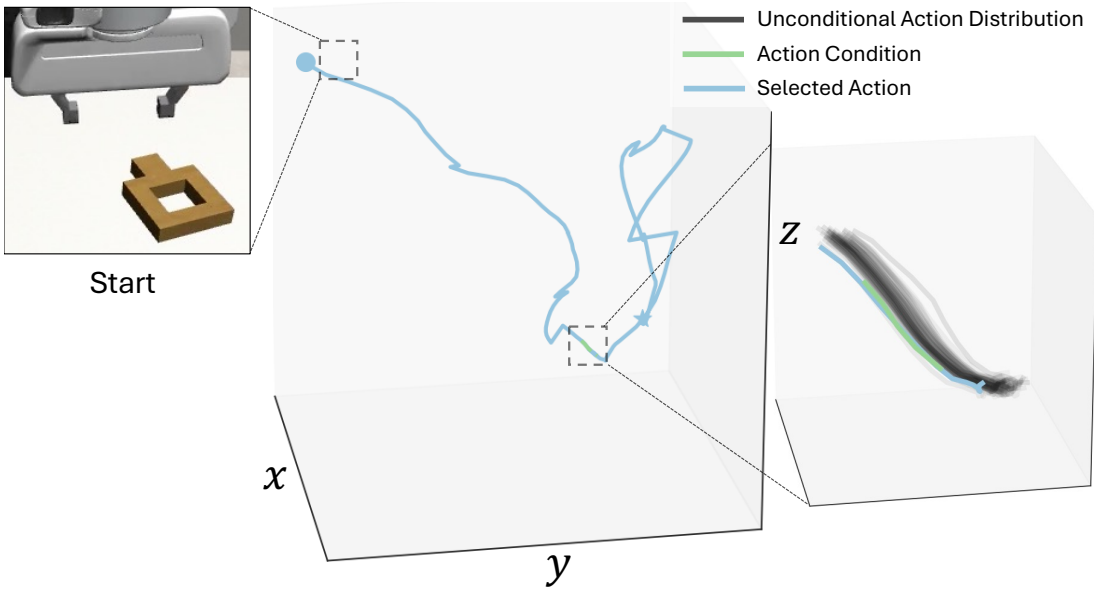
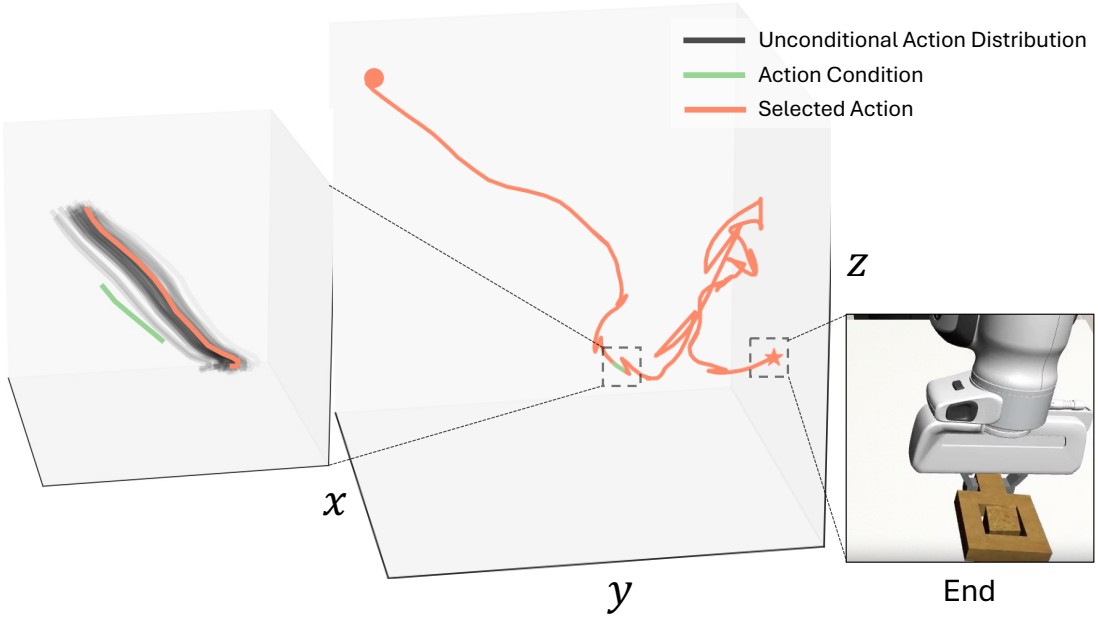


Fig. 15: **Effect of Tracking Error Threshold on Policy Performance.** This figure shows how different tracking error thresholds influence policy performance, measured by success rate (right) and trajectory smoothness (left). Each method is evaluated over 50 scenarios, repeated across three trials. The box plots display the maximum, mean, and minimum values per evaluation. The highest tracking error threshold (0.04) leads to a decline in success rate, while a more conservative threshold improves overall performance.



(a) Successful rollout trajectory with action-conditioning.



(b) Successful rollout trajectory without action-conditioning.

Fig. 16: **Effect of Action-Conditioning on Smoothness of End-Effector Trajectories.** This figure illustrates end-effector trajectories, of a policy rollout on the square task, comparing scenarios with action-conditioning (Figure 16a, blue trajectory) and without action-conditioning (Figure 16b, red trajectory). Snapshots depicting the initial and final states of the square task are provided for each scenario. The guiding action (green line) and sampled unconditional actions (grey lines) are depicted, alongside the selected final action (solid colored lines). With action-conditioning, the chosen action closely aligns with the guided prediction (green), leading to smoother, goal-directed trajectories. Without action-conditioning, the final actions deviate significantly from the guide, resulting in less consistent trajectories. We further provide the snapshots of the initial state and the final state of the square task.

TABLE IV: Key Parameters of Policy Architecture

Parameter	Value
General Settings	
Algorithm	diffusion_policy
Sequence Length	32
Frame Stack	4
Batch Size	128
Num. Epochs	2000
Horizon Settings	
Observation Horizon	4
Action Horizon	32
Prediction Horizon	32
UNet & Diffusion Settings	
UNet Enabled	True
Diffusion Step Embed Dim	256
UNet Down-dimensions	256, 512, 1024
Kernel Size	5
EMA & DDIM	
EMA Enabled	True (power: 0.75)
DDIM Enabled	True
Train Timesteps	100
Inference Timesteps	10
Beta Schedule	squaredcos_cap_v2
Future Action Conditioning	
Enabled	True
Horizon	4
p_{cond}	0.3
Weight	1.0
Null Token	zero
RGB Encoder Settings	
Vision Encoder	ResNet18
Pooling	kp = 32, temp = 1.0
Randomizer	CropRandomizer (116×116, 1 crop)

TABLE V: Optimal Hyperparameter for Consistency Guiding

Task	CFG weight	TEB (Ori.)	TEB (Pos.)
Can	0	0.05	0.02
Lift	1	0.05	0.02
Square	1	0.05	0.02
Mug Cleanup	1	0.05	0.02
Stack	0	0.03	0.01

TABLE VI: Controller and SAIL parameters for each simulated task

	lift	can	square	stack	mug cleanup
K_p	3000	3000	1000	3000	2000
damping	0.5	0.5	1	.75	.75
slowdown c	0.2	0.5	1.0	0.2	0.5

TABLE VII: Controller Gains for Demo Collection and SAIL Execution on Real Robot

	Demo Collection	SAIL Execution
K_p^{pos}	150	300
K_v^{pos}	24.5	34.6
K_p^{rot}	250	400
K_v^{rot}	31.6	40.0

TABLE VIII: Results of Ablation in Sim

		DP [7]	SAIL	-HG	-AS	-C	Commanded Poses
Lift	SR ↑	**1.00**	**1.00**	0.67	0.97	0.98	0.89
	TPR↑	0.46	**1.68**	0.25	1.57	1.58	1.8
	ATR↓	2.23	**0.61**	3.39	0.63	0.63	0.52
	SOD↑	1.08	**3.98**	0.71	3.85	3.84	4.63
Can	SR↑	**0.97**	0.92	0.83	0.95	0.89	0.63
	TPR↑	0.18	0.51	0.18	**0.60**	0.50	0.37
	ATR↓	5.52	1.81	4.36	**1.61**	1.79	1.65
	SOD↑	1.05	3.20	1.33	**3.60**	3.23	3.52
Square	SR↑	0.83	**0.86**	0.59	0.64	0.79	0.31
	TPR↑	0.10	0.13	0.06	**0.25**	0.13	0.04
	ATR↓	7.56	6.41	8.8	**2.50**	5.78	4.25
	SOD↑	0.99	1.18	0.86	**3.01**	1.31	1.77
Stack	SR↑	**1.00**	0.98	0.9	0.94	0.95	0.82
	TPR↑	0.19	**0.66**	0.15	0.61	0.62	0.43
	ATR↓	5.50	**1.56**	6.6	1.71	1.56	2.81
	SOD↑	0.98	**3.47**	0.86	3.15	3.46	1.92
Mug	SR↑	0.68	**0.72**	0.53	0.44	0.68	0.54
	TPR↑	0.03	**0.08**	0.01	0.07	**0.08**	0.03
	ATR↓	17.44	8.09	18.24	**5.37**	8.15	17.38
	SOD↑	0.97	2.09	0.92	**3.14**	2.07	0.92