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

Nadun Ranawaka Arachchige*, **Zhenyang Chen***, **Wonsuhk Jung, Woo Chul Shin, Rohan Bansal, Pierre Barroso, Yu Hang He, Yingyan Celine Lin, Benjamin Joffe, Shreyas Kousik†, Danfei Xu†**
Georgia Institute of Technology, Atlanta, GA, USA
* indicates equal contribution, † indicates equal advising
Correspondence: nadun.ranawaka@gatech.edu

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://nadunranawaka1.github.io/sail-policy/

Keywords: Visuomotor Imitation, Robot Learning Systems, Manipulation

1 Introduction

Speed is essential for real-world robot learning applications. Recent offline imitation learning methods [1, 2] have excelled in complex tasks like deformable object manipulation and non-prehensile actions. However, their performance heavily depends on human demonstrations, which are typically slow, leading policies to inherit sluggish motions. Therefore, this paper addresses the question: *How can we speed up the execution of learned visuomotor policies beyond demonstration speed?* A key challenge is that increased execution speeds alter robot dynamics, introducing tracking errors and dynamic effects, which in turn shifts the observation distribution and further causes the policy to deviate from its prior distribution. Moreover, practical acceleration is constrained by inference la-

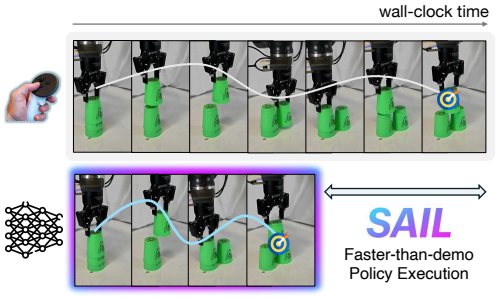


Figure 1: The goal of our system, Speed Adaptation for Imitation Learning (SAIL), is to speed up the execution of a learned visuomotor policy such that the robot can complete manipulation tasks faster than in the original training demonstrations.

tency, sensor delays, and control bandwidth [3], making speed-up a full-stack challenge. As a result, previous works typically assume consistent execution speeds during training and deployment.

We introduce **S**peed **A**daptation for **I**mitation **L**earning (SAIL) as a full-stack framework to tackle this problem (Fig. 1). Our key insight is that successfully accelerating policy execution requires addressing both *changing robot dynamics* and the resulting *state-action distribution shifts*. At the policy level, SAIL features: (1) an controller error-aware generative guidance algorithm that generates temporally-consistent action predictions at varying execution speed (Sec. 4.1); and (2) adaptive speed modulation that dynamically adjusts execution speed based on task demands, slowing down for precision and accelerating otherwise (Sec. 4.3). At the system level, SAIL: (1) trains the policy model to predict controller-invariant action targets and tracks the targets with high-fidelity controller to mitigate dynamic shifts (Sec. 4.2); and (2) adaptively schedules action execution to maintain real-time control despite sensing and computation delays (Sec. 4.4).

This full-stack approach allows SAIL to achieve significantly higher *task throughput* across a variety of manipulation tasks. We validate our key technical insights and design choices through experiments in simulation and demonstrate that SAIL achieves up to a 4$\times$ speedup over demonstration speed while maintaining high task success rates. We also show that, on two physical robot systems with distinctive controller and dynamics, SAIL achieves up to a 3.2$\times$ speedup across 7 tasks involving challenges such as long task horizons, high precision, and bimanual coordination.

2 Related Work

Offline Imitation Learning. Offline imitation learning (LfD) methods are common ways to program robots to learn behaviors from human demonstrations [4, 5, 6, 7]. Particularly, behavior cloning (BC) [7] trains deep networks to map observations to actions from demonstration data. Recent works in BC have introduced deep generative models [2, 1] to preserve the multimodal aspects of real-world trajectories. However, as noted by prior works [2, 8], sampling from these learned trajectory distributions can break temporal dependencies between consecutive prediction steps. This challenge is further exacerbated under higher execution speeds. Relevant to our goal of speed-adaptive imitation are state-space frameworks like Dynamic Movement Primitives (DMPs) [9, 10, 11] and Riemannian Motion Policies (RMPs) [12], which theoretically support temporal modulation of motion. Although a few approaches [13, 14, 15, 16] have incorporated such techniques into end-to-end imitation learning, the field has more widely embraced simpler algorithms like Diffusion Policy [1].

Better-than-Demonstration Imitation Learning. There are multiple lines of work that aim to learn better-than-expert policies [17, 18, 19]. For instance, T-REX [17] learns better-than-demonstration policies by using a ranking-based reward function to evaluate unseen policy behaviors. Others [3, 20] rely on subsequent self-supervised online learning with task-specific reward design [21], enabling a 1.1–1.3$\times$ increase in execution speed at the expense of a loss in success rate. Furthermore, these methods typically operate in an Inverse Reinforcement Learning [22, 23, 24] 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. Some methods, such as VFIL[25], attempt to learn variable speed motions by using user-provided labels whereas our method does not require speed annotations. Recent works, such as SPHINX [26], have shown modest improvements in execution time and speedup. Nevertheless, these gains are typically byproducts of their methods, whereas we explicitly target faster execution as a primary objective.

System Integration for Imitation Learning. Recent works have expanded beyond pure algorithmic innovations to develop full-stack systems for imitation learning [2, 27, 28, 29, 30], with systems such as Mobile-ALOHA [27] and UMI [30] showing tremendous value in full-stack optimization of low-level robot controllers with learning algorithms. However, none of these works consider the problem of deliberately varying the execution speed between demonstration and policy execution. Our core contribution lies in identifying key challenges for this new problem setting and proposing a system that enables faster-than-demonstration policy execution.

3 Preliminaries, Challenges, and Problem Statement

Policy and Controller Hierarchy. We consider a robot control system structured in two levels: (1) a high-level neural network policy $\pi(x, o)$ generating an action command a based on the current robot state x and sensory observation o, and (2) a low-level robot controller $\mathcal{K}$ translating a into robot joint torques u. To make our method generally applicable, we follow a common setup [1, 30, 31] and assume that π outputs action trajectories $a_t = [x_t^{\mathrm{d}}, x_{t+1}^{\mathrm{d}}, \cdots, x_{t+H-1}^{\mathrm{d}}]$, where H is the prediction horizon and each x_i^{d} includes the desired SE(3) end-effector pose and gripper command. The controller $\mathcal{K}$ tracks this trajectory, calculating the torque u_t for each desired pose x_t^{d} given the current state x_t and a fixed time interval δ^* between reference configurations: $u_t = \mathcal{K}(x_t^{\mathrm{d}}, x_t, \delta^*)$. The controller $\mathcal{K}$ typically operates at a significantly higher frequency than the policy inference rate, executing multiple control steps to track each x_t^{d}. Policies are usually deployed in a receding-horizon fashion, executing a portion of the trajectory ($H^{\mathrm{e}} < H$ steps) before replanning.

Offline Imitation Learning. We focus on offline imitation learning, where the policy π is learned from a static dataset $\mathcal{D} = \{(o_t, x_t, x_t^{\mathrm{d}})\}_{t=1}^{n}$ collected via teleoperation. Each datum contains the sensory observation o_t, robot state x_t, and the desired configuration x_t^{d} commanded by the teleoperator, sampled at a fixed time interval δ^*. The action supervision a_t for training is formed by extracting sequences of future desired configurations $[x_t^{\mathrm{d}}, \cdots, x_{t+H-1}^{\mathrm{d}}]$ from the demonstration. In this work, we focus on Diffusion Policy (DP) [1] as our representative high-performance policy model. DP generates multi-step action trajectories via iterative denoising based on the current state and observation (o_t, x_t), following $a_n = a_{n+1} - \gamma\varepsilon_\theta(o_t, x_t, a_{n+1}, n) + \mathcal{N}(0, \sigma_n^2\mathbf{I})$, where ε_θ is the learned denoising network. This action chunking [1, 2, 31] capability is advantageous for handling inference latency. However, finite training data and the probabilistic nature of the policy can lead to temporal inconsistencies between consecutively predicted trajectories [8], potentially causing jerky motion, especially at high speeds. Furthermore, network inference introduces latency that must be managed for real-time, high-speed control (Sec. 4.4). We note that all but one component in our system can apply to other generative policy models, such as ACT [2], which we show in App. J.

Challenges and Problem Statement. Executing learned policies faster than demonstrated introduces critical challenges beyond standard imitation learning. Faster execution drastically exacerbates *distribution shift*, as altered system dynamics and controller errors push the policy into unfamiliar Out-of-Distribution states where compounding errors [32] lead to failure. Simultaneously, achieving speedup strains *execution fidelity*: low-level controllers may struggle to track fast trajectories accurately, the assumption of consistent controller behavior between demonstration and execution breaks down, and tasks inherently require slower speeds during high-precision phases. Finally, real-world *system latencies* impose hard physical limits on control loop frequency, fundamentally constraining maximum achievable speed. *Therefore, our core research problem is*: Given a policy π trained with time interval δ^*, how can we execute it using a faster, time-varying time interval $\delta_t = c_t\delta^*$ (with speedup factor $c_t \leq 1$) to significantly increase successful task throughput while overcoming the intertwined challenges of distribution shift, execution fidelity, and latency?

4 Speed Adaptation for Imitation Learning (SAIL)

4.1 Consistent Action Prediction via Error-Adaptive Guidance

Executing policies significantly faster than demonstrated fundamentally alters controller dynamics. This leads to increased tracking error, pushing the robot state (o_t, x_t) Out-of-Distribution (OOD) compared to its training experience captured in dataset $\mathcal{D}$. This OOD shift poses a critical challenge during receding horizon execution with visuomotor policies. When faced with OOD inputs resulting from tracking errors in the previous execution step, the policy π can produce subsequent action predictions $\hat{a}_{0:H}$ that are inconsistent with the just-executed actions $a_{H^{\mathrm{e}}:H^{\mathrm{e}}+H^{\mathrm{f}}}$. As shown in Fig. 3, this prediction **divergence** between planning steps manifests as jerky or unstable robot motion.

To mitigate this, one might consider enforcing temporal smoothness using Classifier-Free Guidance (CFG) [33]. CFG encourages consistency by conditioning the prediction of the next action

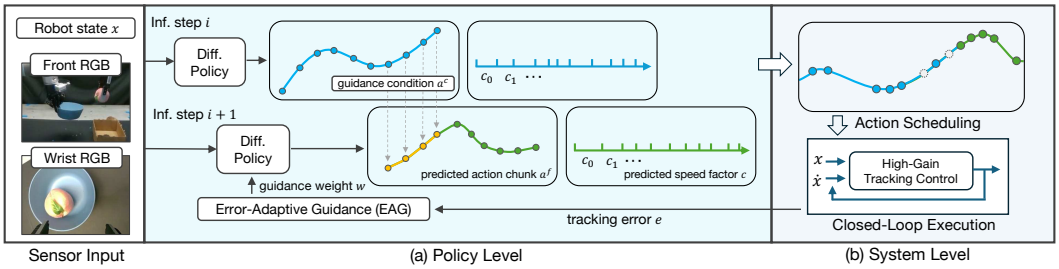


Figure 2: **System Overview.** SAIL operates at two levels: (a) **Policy Level**: Given raw sensor input, the policy generates (1) temporally-consistent action predictions through error-adaptive guidance (EAG) and (2) time-varying speedup factor. (b) **System Level**: The predicted actions are scheduled for execution while accounting for sensing-inference delays, with outdated actions being discarded. The actions are tracked with a high-fidelity controller at the specified time parametrization.

sequence $\hat{a}_{0:H}$ on the tail of the previously planned sequence $a^{\mathrm{c}} = a_{H^{\mathrm{e}}:H^{\mathrm{e}}+H^{\mathrm{f}}}$ (where H^{e} is the number of actions executed from the current chunk before re-planning and H^{f} is the number of actions used for conditioning the next prediction). This is achieved by blending the conditional score estimate $\varepsilon_\theta(a^{\mathrm{f}}, a^{\mathrm{c}} | o_t, x_t)$ with the unconditional one $\varepsilon_\theta(a^{\mathrm{f}}, \emptyset | o_t, x_t)$ using a guidance weight w as $\varepsilon_\theta^{\mathrm{guided}}(a^{\mathrm{f}}, a^{\mathrm{c}} | o_t, x_t) = \varepsilon_\theta(a^{\mathrm{f}}, \emptyset | o_t, x_t) + w(\varepsilon_\theta(a^{\mathrm{f}}, a^{\mathrm{c}} | o_t, x_t) - \varepsilon_\theta(a^{\mathrm{f}}, \emptyset | o_t, x_t))$.

However, naively applying CFG guidance ($w > 0$) during high-speed execution can be detrimental. If significant tracking error occurred while executing a^{c}, the input (x_t, a^{c}) to the conditional model $\varepsilon_\theta(\cdot, a^{\mathrm{c}} | o_t, x_t)$ is effectively OOD. Conditioning strongly on this unreliable information provides misleading guidance, potentially further exacerbating divergence. This challenge highlights the limitations of approaches that either perform smoothing *post-hoc* after action generation [8, 2] or implicitly assume consistent execution dynamics [1, 34] without directly addressing the OOD inputs caused by controller shifts during the generation process itself.

Therefore, we propose **Error-Adaptive Guidance (EAG)**. We recognize that the utility of conditioning via CFG depends on the quality of the conditioning signal, which degrades with increasing tracking error. We use the current end-effector tracking error $e = \mathrm{error}(x^{\mathrm{d}}, x^{\mathrm{current}})$ as an efficient proxy for this potential OOD state (validated in Appendix H.3). Our algorithm dynamically adjusts the CFG guidance weight w: if tracking is accurate ($e \leq \rho$), we apply standard CFG guidance ($w > 0$) assuming reliable conditioning. Conversely, if the error is large ($e > \rho$), indicating the conditioning input a^{c} is likely unreliable due to the OOD state, we disable guidance ($w = 0$) and rely solely on the unconditional prediction $\varepsilon_\theta(a^{\mathrm{f}}, \emptyset | o_t, x_t)$. This adaptive approach allows SAIL to enforce temporal consistency when tracking is accurate, while preserving robustness by avoiding potentially incorrect guidance caused by controller shift. We illustrate how EAG is integrated with the overall system in Fig. 2 and detail its implementation in Alg. 2.

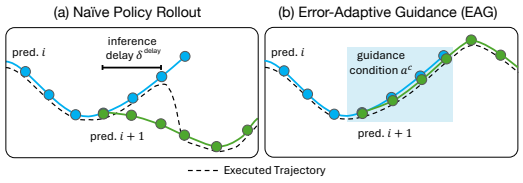


Figure 3: **Divergence during receding horizon execution.** We found that the naive policy rollout occasionally produces inconsistent predictions between planning iterations, as shown in (a). For example, the blue and green trajectories are two consecutive trajectories that diverge in path. This can cause a jerky executed trajectory (black dashed line) during receding horizon control. EAG addresses this problem by enforcing consistent consecutive predictions via conditional guidance a^c.

4.2 Reducing Controller Shift via Controller-invariant Action Target

While Sec. 4.1 adaptively handles Out-of-Distribution (OOD) states algorithmically, a primary source of this distribution shift during high-speed execution is the changing behavior of the low-level controller. Standard Imitation Learning often trains policies π to predict the **commanded** poses x^{d} from teleoperation (collected at interval δ^*, often with a low-gain controller to ensure

smooth user experience) and uses the same teleoperation controller $\mathcal{K}^{\text{teleop}}$ for execution. However, running $\mathcal{K}^{\text{teleop}}$ at a faster interval $\delta_t < \delta^*$ alters its dynamics and tracking error profile, providing OOD state inputs to the policy and hindering performance (Fig. 4, Middle).

To mitigate this at the system level, we first change *what* the policy predicts. Instead of the commanded pose x^{d}, we train π to predict the **reached pose** x directly from the demonstration data (Fig. 4, Right). The reached pose represents the robot's actual trajectory and is inherently achievable. Crucially, this target is largely *invariant* to the specific dynamics of the potentially compliant or noisy $\mathcal{K}^{\text{teleop}}$ used during data collection, providing a more stable prediction target for the policy across different execution speeds. Complementing this, we also change *how* the predicted action is executed. During high-speed deployment, we replace the original $\mathcal{K}^{\text{teleop}}$ with a dedicated, *high-fidelity tracking controller* $\mathcal{K}^{\text{e}}$ optimized for fast and accurate motion. This controller takes the policy's predicted reached poses x_t as targets and generates torques $u_t = \mathcal{K}^{\text{e}}(x_t, x^{\text{current}}, \delta_t)$ to follow them closely, even at the reduced interval δ_t. Using a high-performance controller (like our high-gain OSC detailed in App. D or potentially MPC) to track the invariant reached pose target effectively decouples policy execution from the variable dynamics of the teleoperation controller, reducing controller-induced distribution shift.

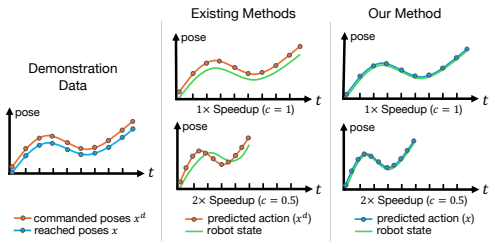


Figure 4: **Commanded vs Reached Pose.** (Left) Teleoperator commands x^{d} to the robot, and the robot reaches poses x. (Middle) Most policies are trained to predict x^{d} and suffer from error profile change during speeding up execution. (Right) We minimize this shift by training policies to predict the reached poses x and track these with a high-fidelity controller.

4.3 Adaptive-Speed Policy Execution

Executing all parts of a manipulation task at maximum speed can compromise success, especially during precise interactions like grasping or alignment. SAIL automatically identifies such *critical actions* and dynamically adjusts the instantaneous speedup factor c_t based on the real-time context, slowing down during critical actions and speeding up otherwise.

We identify critical actions using two complementary methods based on analyzing demonstration data and runtime predictions. (1) *Motion Complexity Analysis*: Inspired by [35], we perform offline analysis (e.g., DBSCAN clustering [36] on demonstration waypoints) to flag segments with high geometric complexity. We then train the policy π to predict, alongside its action, whether the current action corresponds to a phase that requires complex and careful motion. (2) *Gripper Event Detection*: A simple heuristic to identify likely interaction phases at runtime is detecting changes (opening/closing) in the predicted gripper state within the policy's output action sequence $a_{t:t+H}$. Both of these methods yield a binary critical action flag $k_t \in \{0,1\}$ for the current timestep t, where $k_t = 1$ indicates a critical action. We modulate the speedup factor between preset slow (c^{slow}) and fast (c) values: $c_t = k_t \cdot c^{\text{slow}} + (1 - k_t) \cdot c$. This allows SAIL to automatically slow down for precision and speed up during simpler motions like reaching. We include more details in App. F.

4.4 Maintaining Real-Time Control at High Speed Under System Latency

Action scheduling. While adaptive modulation determines the *desired* execution speed c_t, achieving stable control at high average speeds requires explicitly handling inherent system latencies [30]. There exists an irreducible sensing-to-action delay δ^{delay} between requesting sensor data (at t^o) and receiving the computed action sequence from the policy (at t^a). To maintain continuous robot motion and prevent pauses, SAIL executes actions planned from the *previous* policy inference step during this δ^{delay} interval. Once the new action sequence $\hat{a}_{0:H}$ arrives at t^a, the system schedules its execution using the adaptively determined interval $\delta_t = c_t \delta^*$, seamlessly transitioning from the old plan by discarding any actions from the previous plan scheduled after t^a.

Upper bounding speedup. Crucially, the latency δ^{delay} imposes a fundamental physical limit on the maximum achievable average speedup. To prevent "action exhaustion"—running out of commands before the next inference cycle completes—the actual execution interval δ_t must be lower-bounded. This lower bound, $\delta^{\text{lb}} > \delta^{\text{delay}}/(H^{\text{p}} - H^{\text{c}})$ (where H^{p} is prediction horizon and H^{c} is conditioning length, derived in App. E), represents the minimum time required per step to sustain continuous operation. Therefore, the adaptively chosen desired interval $c_t\delta^*$ must respect this physical limit. The final execution interval used by the controller at time t is $\delta_t = \max(c_t \cdot \delta^*, \delta^{\text{lb}})$. This ensures SAIL dynamically varies speed according to c_t, but never exceeds the system's physical capacity.

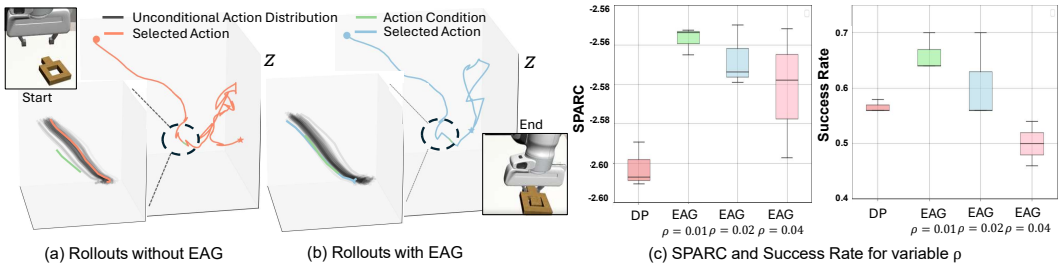
5 Evaluations

We evaluate SAIL to test the following hypotheses: ***H1***: Error-Aware Guidance (EAG) generates temporally-consistent actions that improve policy performance at high speed, ***H2***: All components of SAIL are critical for enabling faster-than-demo execution while keeping a high success rate, and ***H3***: SAIL is generally deployable to physical robots on realistic tasks.

Metrics. Our primary evaluation metric is throughput-with-regret (TPR↑), which rewards faster successes while penalizing all failures equally, thus reflecting a policy's capability to maintain success rates at higher execution speeds. We further consider two suites of metrics. The first suite focuses on task performance and efficiency, which includes task success rate (SR↑), TPR, average time for successful rollouts (ATR↓), speedup-over-demo (SOD↑). The second suite characterizes the generated motion trajectories (e.g., smoothness), including consistency (CON↓), spectral arc length [37] (SPARC↑), log dimensionless jerk [37] (LDLJ↓), and weighted Euclidean distance [8] (WED↓). Detailed metric descriptions are in App. B.

5.1 Simulation Evaluation: SAIL Achieves High Task Throughput

Setup. We evaluate SAIL and a variety of baselines on standard manipulation benchmarks from RoboMimic [38] and MimicGen [39]: Lift, Can, Square, Stack, and Mug Clean-up (details in App. C). Since few imitation learning methods directly address execution speedup, we design several baselines for comparison, including AWE [35] which achieves speedup as a side effect of its waypoint-based approach. Our baselines are: (1) **DP** [1]: Executes actions at the original demonstration speed, serving as our primary baseline. (2) **DP-Fast**: Executes Diffusion Policy actions at an accelerated fixed frequency using a low-gain controller, representing the naive approach to speedup. (3) **Aggregated Actions**: Operates in delta Cartesian Space by aggregating consecutive actions in similar directions (detailed in App. I). (4) **AWE** [35]: Uses automatically extracted waypoints to generate absolute action labels. (5) **BID-Fast**: the same as **DP-Fast** but using BID [8] to enforce consistency between consecutive predictions. See the architectures and parameter details in App. I.



(a) Rollouts without EAG (b) Rollouts with EAG (c) SPARC and Success Rate for variable ρ

Figure 5: We show that EAG generates temporally-consistent motion, comparing sample rollouts without EAG in (a) and rollouts with EAG in (b). We further illustrate how the success rate and smoothness of trajectories are affected by the error threshold ρ in (c).

EAG generates temporally-consistent motion at high speeds (*H1*). To assess the effectiveness of EAG, we evaluate both trajectory smoothness and task performance. Qualitative motion results and quantitative metric (SPARC) in Fig. 5 shows that EAG significantly enhances motion consis-

tency, which in turn improves task success rates during accelerated execution (SR). This is further confirmed by the ablation study **SAIL(-C)** in Table J.6, where removing consistency-preserving trajectory generation reduces performance. Detailed results presented in App. H.1 confirm the effectiveness of EAG and the importance of the adaptive tracking error cutoff for conditioning.

SAIL achieves much higher throughput than baselines (*H1*). As seen in Table 1, SAIL can achieve up to 3× throughput of baselines such as DP [1] 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 Fig. 6. We observe that SAIL is able to smoothly improve the TPR as c_t increases, with up to $c_t = 0.1$ (10× speedup) showing its robustness at least in an ideal simulated environment.

Component Ablation (*H2*). We present the conclusion of ablation studies evaluating key components in SAIL, with full results in the Appendix. First, high-gain controllers, while necessary for fast tracking, are sensitive to reference trajectory noise. As shown in Fig. G.5 in App. G.2, noisy references drastically reduce success rates for high-gain control, highlighting the need for smooth references generated by mechanisms like EAG. Second, imitating reached poses is critical for high-speed execution. Substituting commanded poses significantly degrades performance, particularly when replaying demonstrations faster or with higher gains (details in App. G.1). This degradation is quantified in Table J.6: using commanded poses instead of reached poses results in a 55% drop in success rate for **Square** and reduces the average TPR by 0.08 across tasks compared to full SAIL. Finally, adaptive speed modulation (AS) proves vital, especially for precise tasks. Comparing SAIL with **SAIL(-AS)** (SAIL without adaptive speed) in Table J.6, we observe significantly lower success rates without AS, particularly on **Square** and **Mug** (degradations up to 31%). This underscores the importance of dynamically adjusting speed, especially for high-precision scenarios like **Square**.

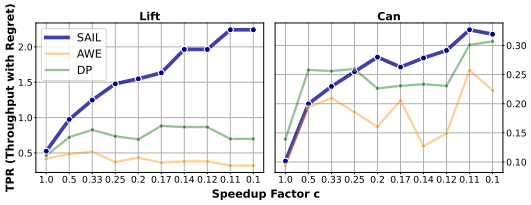


Figure 6: **TPR vs. Speedup Factor on Can and Lift Tasks.** We show that as the speedup factor increases, SAIL's throughput-with-regret increases more than the AWE and DP baselines. In other words, SAIL is able to accumulate more task successes more quickly while limiting task failures.

Table 1: Results of evaluation in simulation (Robomimic [38] and MimicGen [39])

	Lift				Can				Square				Stack				Mug			
Method	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑
DP [1]	**1.00**	0.46	2.23	1.08	**0.97**	0.18	5.52	1.05	0.83	0.10	7.56	0.99	**1.00**	0.19	5.50	0.98	0.68	0.03	17.44	0.97
DP-Fast	0.95	1.02	1.52	1.59	0.87	0.37	2.34	2.48	0.55	**0.15**	**3.42**	**2.20**	0.98	0.44	2.37	2.28	0.56	0.05	9.67	1.74
AWE [35]	**1.00**	0.44	2.35	1.02	0.96	0.17	5.80	1.00	0.83	0.10	8.13	0.93	0.98	0.11	9.01	0.60	**0.75**	0.02	28.79	0.59
Agg. Act.	0.91	0.52	1.78	1.37	0.82	0.16	4.77	1.22	0.29	0.03	4.81	1.57	0.82	0.17	6.18	0.87	0.59	0.03	15.86	1.06
BID-Fast [8]	0.86	0.91	0.97	2.50	0.79	0.34	2.39	2.34	0.49	0.12	3.45	2.18	0.99	0.47	2.61	2.07	0.62	0.06	8.71	1.94
SAIL(Ours)	**1.00**	**1.68**	**0.61**	**3.98**	0.92	**0.51**	**1.81**	**3.20**	**0.86**	0.13	6.41	1.18	0.98	**0.66**	**1.56**	**3.47**	0.72	**0.08**	**8.09**	**2.09**

5.2 Real-World Evaluation: SAIL Achieves High Throughput on Hardware

Table 2: Real-World Evaluation

	Stacking Cups				Wiping Board				Baking				Folding Cloth			
Method	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑
DP-Fast	0.10	-2.28	**14.00**	**1.85**	**0.90**	**3.48**	14.54	2.34	0.90	3.06	16.15	2.26	0.10	-2.28	14.60	2.08
SAIL	**0.40**	**-0.12**	14.71	1.76	0.70	3.18	**10.44**	**3.26**	**1.00**	**4.20**	**14.39**	**2.54**	**0.30**	**-0.78**	**13.68**	**2.22**

	Plate Fruits				Pack Chicken				Bimanual Serve			
Method	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑
DP-Fast	0.60	2.22	13.74	1.66	0.40	0.51	17.33	1.25	0.40	1.00	12.01	1.43
SAIL	**0.80**	**5.46**	**8.53**	**2.67**	**0.90**	**5.22**	**9.40**	**2.30**	**0.70**	**5.40**	**7.19**	**2.39**

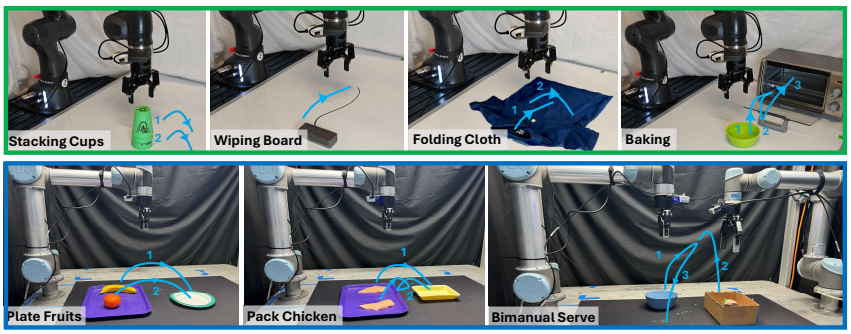


Figure 7: Real-world task setup with two robot platforms (Franka and UR5).

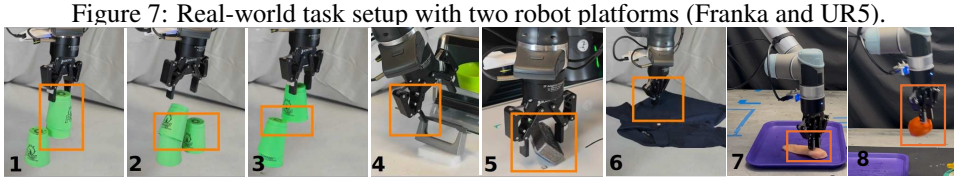


Figure 8: **Commonly-seen failure modes from real-world evaluation.** Speeding up policy execution poses challenges unseen in normal speed execution, which include imprecise grasping (grasping two cups in 1, missing eraser in 5, and missing chicken in 7), low-fidelity tracking (colliding with other cups in 2 and 3, missing handle in 4, missing collar in 6), jerky motion (fruit drops in 8). SAIL effectively reduces such failures under speeding-up execution, leading to higher task throughput.

Setup. We evaluate our method on two different robot platforms (Franka Emika Panda and Bimanual UR5) across 7 challenging manipulation tasks (shown in Fig. 7), including long task horizon, high-precision steps, and bimanual coordination (detail in App. D.3). Both robot platforms run high-fidelity tracking controllers (detail in App. D). We compare **SAIL** ($5\times$ speedup) with **DP-Fast** ($5\times$ speedup). Each method is evaluated for 10 rollouts per task.

Results: SAIL overcomes DP-Fast failure modes (*H3*). Across differences in control systems, robot dynamics, and tasks, SAIL generally improves task throughput. As shown in Table 2, throughput-with-regret (TPR) and SOD both improved on 6/7 challenging tasks, demonstrating SAIL's consistent speed advantage over the baseline sped-up diffusion policy. Qualitatively (Sec. 5.2), 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 $5.4\times$ higher throughput and approximately $1.8\times$ 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. We provide additional discussion in the Limitation section.

6 Conclusion

We formalized and identified challenges in the novel problem of faster-than-demonstration execution of visuomotor policies. Our framework, SAIL, tackles the full-stack problem by combining Error-Adaptive Guidance, controller-invariant targets, adaptive speed modulation, and latency-aware scheduling. Experiments show SAIL achieves up to $4\times$ speedup in simulation and $3.2\times$ speedup in real-world while maintaining high success rates across diverse tasks.

7 Limitations

Through formalizing and proposing a solution for the novel problem of faster-than-demonstration execution, we have identified several fundamental challenges that open up new research directions for the robotics community. First, SAIL focuses on addressing the observation-action drift as a result of controller dynamics shift and does not explicitly tackle the dynamics shift of robot-object interaction. This manifests most clearly in manipulation tasks where object-robot dynamics become significantly more complex at higher speeds—for instance, we observed that in the simulated Can task, increased execution speed can cause the robot to inadvertently throw the can out of the workspace due to increased momentum. Relatedly, the finite-data nature of offline imitation learning makes it vulnerable to distributional shift that cannot be addressed solely by improving the policy learning algorithms. Future research could address this by developing methods to incorporate explicit dynamics modeling into policies, either by leveraging known dynamics models or learning from simulation-based data during training. Additionally, certain tasks such as Square are not amenable to speedup due to precise and complex nature of the task motion, with significant time spent at regular speed for alignment and insertion, reducing the apparent TPR gain as seen in Table 1.

As the field continues to advance learning-based manipulation in the wild, we believe a key focus should be to enable the robot learning system to co-optimize the low-level control and the anticipated dynamic effects of the predicted actions at different execution speeds. We hope this work can open new pathways and facilitate wider adoption of learned policies in real industrial applications.

Acknowledgments

The authors would like to acknowledge the State of Georgia and the Agricultural Technology Research Program at Georgia Tech for supporting the work described in this paper. We also acknowledge funding from the AI Manufacturing Pilot Facility project under Georgia Artificial Intelligence in Manufacturing (Georgia AIM) from the U.S. Department of Commerce Economic Development Administration, Award 04-79-07808, NSF CCF program, Award 2211815, and NSF Award 1937592.

References

[1] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[2] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[3] S. Sakaino, K. Fujimoto, Y. Saigusa, and T. 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.

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

[5] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[6] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

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

[8] Y. Liu, J. I. Hamid, A. Xie, Y. Lee, M. Du, and C. Finn. Bidirectional Decoding: Improving Action Chunking via Closed-Loop Resampling. *arXiv preprint arXiv:2408.17355*, 2024.

[9] A. J. Ijspeort, J. Nakanishi, and S. Schaal. Learning rhythmic movements by demonstration using nonlinear oscillators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 958–963, 2002. doi:10.1109/IRDS.2002.1041514.

[10] A. Ijspeert, J. Nakanishi, and S. Schaal. Trajectory formation for imitation with nonlinear dynamical systems. In *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, volume 2, pages 752–757 vol.2, 2001. doi:10.1109/IROS.2001.976259.

[11] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel. Dynamic movement primitives in robotics: A tutorial survey. *CoRR*, abs/2102.03861, 2021. URL https://arxiv.org/abs/2102.03861.

[12] N. D. Ratliff, J. Issac, and D. Kappler. Riemannian motion policies. *CoRR*, abs/1801.02854, 2018. URL http://arxiv.org/abs/1801.02854.

[13] S. Bahl, M. Mukadam, A. Gupta, and D. Pathak. Neural dynamic policies for end-to-end sensorimotor learning. *Advances in Neural Information Processing Systems*, 33:5058–5069, 2020.

[14] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Control, planning, learning, and imitation with dynamic movement primitives. In *Workshop on Bilateral Paradigms on Humans and Humanoids: IEEE International Conference on Intelligent Robots and Systems (IROS 2003)*, pages 1–21, 2003.

[15] M. Xie, A. Handa, S. Tyree, D. Fox, H. Ravichandar, N. D. Ratliff, and K. V. Wyk. Neural geometric fabrics: Efficiently learning high-dimensional policies from demonstration. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=GTyBkq36tjx.

[16] S. Bahl, A. Gupta, and D. Pathak. Hierarchical neural dynamic policies. *CoRR*, abs/2107.05627, 2021. URL https://arxiv.org/abs/2107.05627.

[17] D. Brown, W. Goo, P. Nagarajan, and S. Niekum. Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations. In *International conference on machine learning*, pages 783–792. PMLR, 2019.

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

[19] Y.-H. Wu, N. Charoenphakdee, H. Bao, V. Tangkaratt, and M. Sugiyama. Imitation learning from imperfect demonstration. In *International Conference on Machine Learning*, pages 6818–6827. PMLR, 2019.

[20] Y. Saigusa, A. Sasagawa, S. Sakaino, and T. 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.

[21] Y. Saigusa, S. Sakaino, and T. Tsuji. Imitation learning for nonprehensile manipulation through self-supervised learning considering motion speed. *IEEE Access*, 10:68291–68306, 2022.

[22] A. Y. Ng, S. Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[23] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

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

[25] N. Masuya, S. Sakaino, and T. Tsuji. Variable-frequency imitation learning for variable-speed motion, 2024. URL https://arxiv.org/abs/2411.12310.

[26] P. Sundaresan, H. Hu, Q. Vuong, J. Bohg, and D. Sadigh. What's the Move? Hybrid Imitation Learning via Salient Points. *arXiv preprint arXiv:2412.05426*, 2024.

[27] Z. Fu, T. Z. Zhao, and C. Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

[28] X. Cheng, J. Li, S. Yang, G. Yang, and X. Wang. Open-television: Teleoperation with immersive active visual feedback. *arXiv preprint arXiv:2407.01512*, 2024.

[29] A. Iyer, Z. Peng, Y. Dai, I. Guzey, S. Haldar, S. Chintala, and L. Pinto. Open teach: A versatile teleoperation system for robotic manipulation. *arXiv preprint arXiv:2403.07870*, 2024.

[30] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song. Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots. In *Proceedings of Robotics: Science and Systems (RSS)*, 2024.

[31] S. Liu, L. Wu, B. Li, H. Tan, H. Chen, Z. Wang, K. Xu, H. Su, and J. Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

[32] S. Ross, G. Gordon, and D. 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.

[33] J. Ho and T. Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[34] S. H. Høeg, Y. Du, and O. Egeland. Streaming diffusion policy: Fast policy synthesis with variable noise diffusion models. *arXiv preprint arXiv:2406.04806*, 2024.

[35] L. X. Shi, A. Sharma, T. Z. Zhao, and C. Finn. Waypoint-Based Imitation Learning for Robotic Manipulation. In *Conference on Robot Learning*, pages 2195–2209. PMLR, 2023.

[36] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. KDD'96, page 226–231. AAAI Press, 1996.

[37] 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.

[38] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What Matters in Learning from Offline Human Demonstrations for Robot Manipulation. In *5th Annual Conference on Robot Learning*, 2021.

[39] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox. MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations. In *7th Annual Conference on Robot Learning*, 2023.

[40] 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.

[41] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu. robosuite: A Modular Simulation Framework and Benchmark for Robot Learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[42] E. Todorov, T. Erez, and Y. 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.

[43] Y. Zhu, A. Joshi, P. Stone, and Y. Zhu. VIOLA: Imitation Learning for Vision-Based Manipulation with Object Proposal Priors. *arXiv preprint arXiv:2210.11339*, 2022. doi:10.48550/arXiv.2210.11339.

[44] S. Scherzinger, A. Roennau, and R. Dillmann. Forward dynamics compliance control (fdcc): A new approach to cartesian compliance for robotic manipulators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4568–4575, 2017. doi:10.1109/IROS.2017.8206325.

[45] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

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

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

[48] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

A Table of Contents

The Appendix contains the following content:

- **Formulas and explanation of evaluation metrics** (Appendix B): this section explains the different metrics used in the experiments for this paper.
- **Simulation experiment setup** (Appendix C): this section describes the task suite and simulator used for the simulation experiments, as well as specific changes we made to enable speedup.
- **Real world setup and tasks** (Appendix D): we detail our hardware and data collection setup and provide descriptions of the different tasks we used to evaluate SAIL.
- **Derivation of Action Interval** (Appendix E): we derive a theoretical bound for the speedup factor c_t that still maintains continuous execution without pauses due to inference.
- **Modulating speed** (Appendix F): this section describes two different methods of detecting critical actions that allow SAIL to modulate the speedup factor c during various stages of a task.
- **Experiments for justifying components of SAIL** (Appendix G): in this section we present mini experiments that give insights into different components of SAIL and how they affect performance.
- **Error adaptive guidance experiments** (Appendix H): we present a deep dive into EAG and the importance of varying guidance based on tracking error.
- **Experiment Parameters and Baseline Implementation** (Appendix I): this section lists out the different hyperparameters used for the policies and robot controllers and the implementation of the **Aggregated Actions** baseline in Table 1.
- **Ablation and Action Chunking with Transformers results** (Appendix J) : we perform a detailed ablation of the different components of SAIL and present results using a different policy algorithm - ACT [2].

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:

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

   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, $\text{CON} = \hat{a}_{H^{\text{f}}} - a_{H^{\text{e}}+H^{\text{f}}}$, following the notation of Sec. 4.1.

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 [37]. It is an extended version of Spectral Arc Length (SAL) [40].
   In SAL, the magnitude spectrum $V(\omega)$ of the Fourier transform of a speed profile v_t is normalized by its DC value $V(0)$:
   $$\hat{V}(\omega) = \frac{V(\omega)}{V(0)} \tag{2}$$
   SAL integrates the arc length of $\hat{V}(\omega)$ over frequencies from 0 up to a cutoff frequency ω_c:
   $$\text{SAL} \triangleq -\int_0^{\omega_c} \left[\left(\frac{1}{\omega_c} \right)^2 + \left(\frac{d\hat{V}(\omega)}{d\omega} \right)^2 \right]^{\frac{1}{2}} d\omega \tag{3}$$
   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:
   $$\omega_c \triangleq \min\Big\{ \omega_c^{\max}, \min\big\{ \omega, \hat{V}(r) < \overline{V} \ \ \forall r > \omega \big\} \Big\}. \tag{4}$$
   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 [37]:
   $$\text{LDLJ} \triangleq -\ln \left| -\frac{(t_2 - t_1)^5}{(v^{\text{peak}})^2} \int_{t_1}^{t_2} \left| \frac{d^2 v_t}{dt^2} \right|^2 dt \right| \tag{5}$$
   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 [8] to quantify the consistency between overlapping action segments.

C Simulation Experiment Details

C.1 Robot Control and Dynamics Considerations

We control robots in simulation using an OSC controller that takes absolute pose commands, except for the **aggregated actions** baseline, which uses 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 I.4, we list the optimal controller and the upper bound of c in adaptive speed modulation that SAIL uses in the simulated tasks.

C.2 Simulator and Data

We use Robosuite [41] to simulate robots and their environments. Robosuite is built on Mujoco [42] 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 [38] suite of tasks and two tasks from MimicGen [39]. 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.

C.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.

D.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 Sec. 3.

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 [43]. 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

$$\tau = J^\top M(K_p e_p + K_v e_v), \tag{6}$$

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 differentiating a spline over the predicted action trajectory.

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.

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

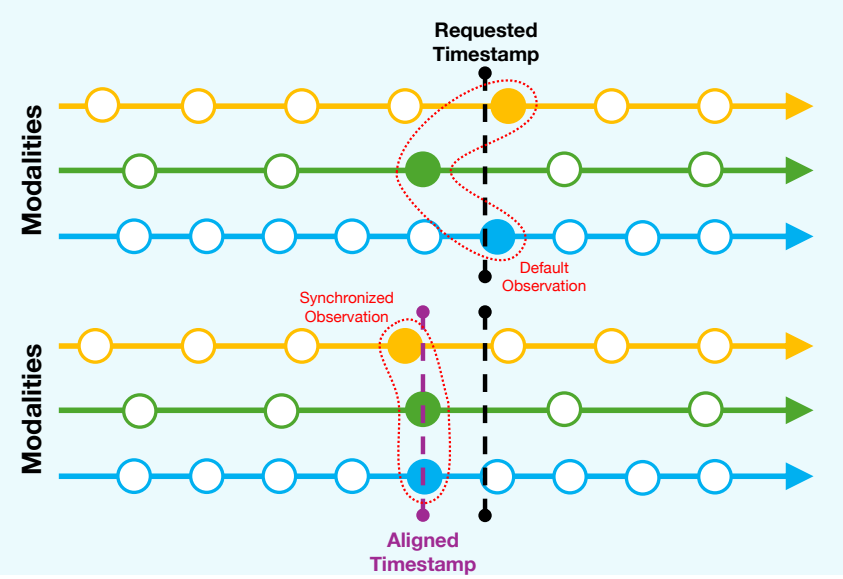


Figure D.1: **Timestamps alignment**. For each modality, we identify the nearest observation (filled circles) to construct the default observation (red dotted outline) at the requested timestamp (black dashed line). Among these nearest timestamps, we determine the farthest one (purple dashed line). We then align the other modalities (filled circles) to this timestamp to obtain the synchronized observation (red dotted outline).

comfortably before resuming the task. Demonstrations are recorded at 20Hz. We collect the robot's *reached poses* and *commanded poses* at 20Hz. Images are attained from the Kinect camera which is collected at 30Hz, while Zed camera is collected at 60Hz. To handle this mismatch, we cache the most recent 100 messages per modality and perform an observation alignment process. Specifically, upon receiving a request for observation at time t, we find the closest timestamps in each cache, then among these we identify the one farthest from t, and realign all modalities to that time to ensure consistency (see Fig. D.1).

We collect 50 demonstrations for each task on the Franka robot.

D.2 UR5 Robot

We designed a modular control and data collection framework for UR5 robots equipped with Cartesian controllers [44] based on ROS 2[1].

Similar to the Franka setup, demonstrations are collected through a teleoperation loop running at 20Hz, while our observation modalities operate at different frequencies: the on-wrist Intel RealSense D405 and static D435 scene cameras run at 90Hz, the robot controllers operate at 125Hz, and the VR device at 72Hz. Our UR5 setup can be used to perform bimanual tasks with a single operator/policy controlling both arms. Each arm's speed can be individually controlled as needed to complete the task.

By scheduling actions our system can execute the learned policies at configurable rates. By default, we match the 20Hz demonstration rate, but the frequency can be dynamically adjusted based on model predictions to speed up or slow down motions.

For the tasks tested on the UR5 robots, we varied the number of demos according to the task length and difficulty.

[1] https://docs.ros.org/en/jazzy/index.html

D.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 Derivation 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} (Sec. 4.1). 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

$$\begin{aligned} & H^{\text{p}} \cdot \delta^{\text{lb}} > \delta^{\text{delay}} + H^{\text{c}} \delta^{\text{lb}} \\ \implies \quad & \delta^{\text{lb}} > \frac{\delta^{\text{delay}}}{H^{\text{p}} - H^{\text{c}}}. \end{aligned} \tag{7}$$

We illustrate this relationship in Fig. E.2. 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 [45]. 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:

$$c_t \in \left(\delta^{\text{lb}} / \delta^*, 1 \right]. \tag{8}$$

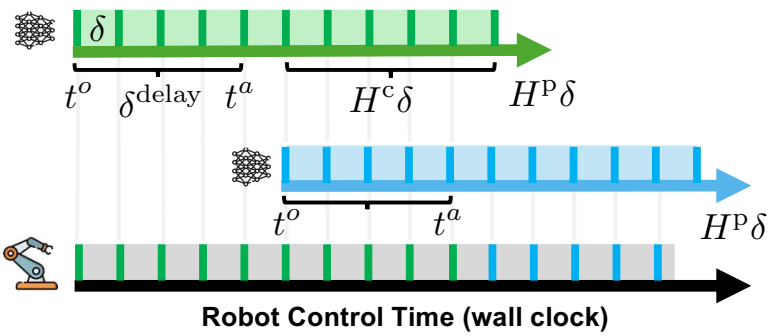


Figure E.2: **Handling latencies in the control loop.** We illustrate the control loop timeline of SAIL and how it handles system latency. The green timeline (top) shows the first action sequence generated at t^o. The sequence spans $H\delta$ with the last H^{c} steps conditioning the next prediction. The blue timeline (middle) shows the next action prediction starts while the system continues to execute the first prediction. The bottom timeline shows the actual robot execution timeline. The system smoothly transitions from the first sequence (green) to the next (blue) without pausing.

Moreover, note that c_t from Sec. 4.3 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 Adaptive Speed Modulation

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.

We propose two techniques to identify critical actions from the demonstration data: measuring motion complexity and using gripper open/close actions. 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:

$$c_t = k_t \cdot c^{\text{slow}} + (1 - k_t) \cdot c. \tag{9}$$

where $c^{\text{slow}} \in (0,1]$ is a slower speedup factor used for critical actions and $c \in (0,1]$ is used otherwise. Note that $c^{\text{slow}} > c$ since the *reciprocal* of the speedup factor determines the speedup. We set c^{slow} and c to empirically-validated presets for each task. Also note that we lower-bound c_t in App. E. A policy rollout using this technique of adaptive speed modulation is shown in Fig. F.3.

Identifying Critical Actions via Motion Complexity. Inspired by Automatic Waypoint Extraction (AWE) [35], 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 [36], 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.

We describe the algorithm to classify critical actions via motion complexity in Alg. 1.

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

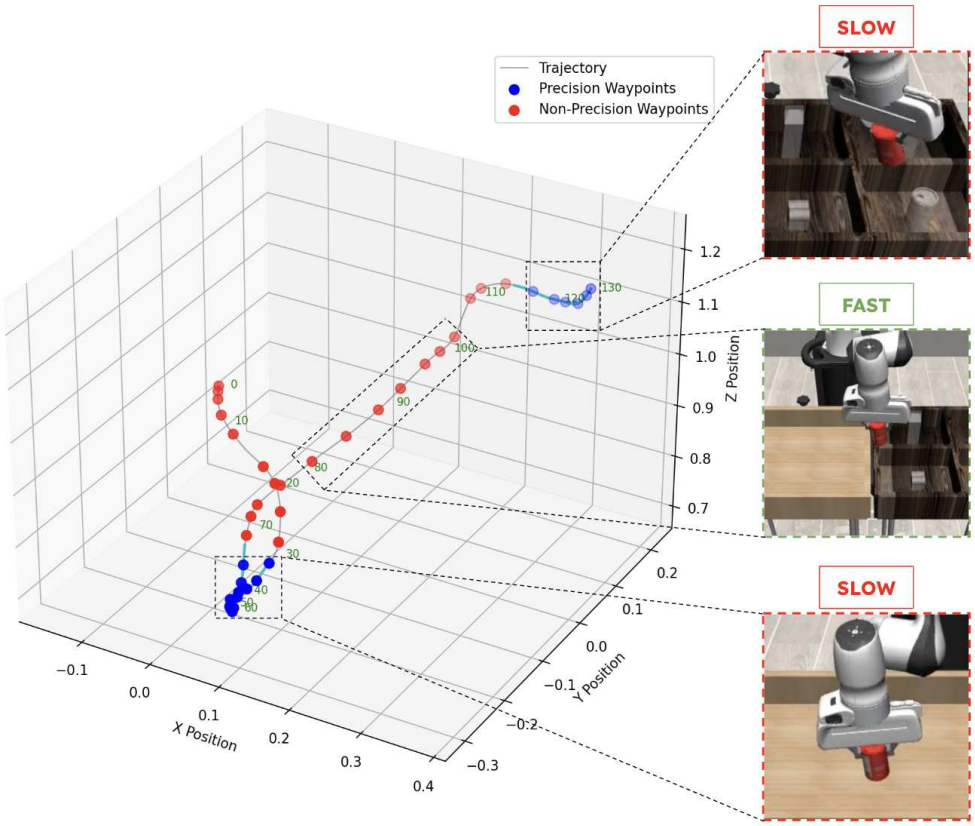


Figure F.3: **Policy rollout with adaptive speed modulation** . Waypoints (red) are generated by [35] given the trajectory as input along with an error threshold. Areas of complex motion (blue), are marked by performing a spatial clustering of the extracted waypoints. Any waypoint outside the cluster 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).

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 for Each Component

G.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 Fig. G.4, **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.

G.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

Algorithm 1: Identify Critical Actions via Motion Complexity

Input: Absolute action sequence $\mathcal{A} = \langle a_1, a_2, \ldots, a_N \rangle$

Function AWE ($\mathcal{A}$, τ) **:**
  return waypoint set $\mathcal{W} = \{w_1, \ldots, w_M\}$ with $w_k = (x_k, y_k, z_k)$, threshold τ;
Function $\mathcal{W}$**:**
  Run DBSCAN on $\{(x_k, y_k, z_k)\}_{k=1}^{M}$ with parameters $(\varepsilon, minPts)$;
  for $k \leftarrow 1$ **to** M **do**
    if w_k is assigned to a cluster **then label**$[k] \leftarrow 1$;
    else label$[k] \leftarrow 0$;
  return label vector **label**;

$\mathcal{W} \leftarrow$ AWE ($\mathcal{A}$, τ) ;
label $\leftarrow \mathcal{W}, \varepsilon, minPts$;
$\mathbf{s} \leftarrow \mathbf{0}_N$;
foreach *consecutive pair* (w_k, w_{k+1}) **do**
  $i \leftarrow \text{index}(w_k)$;
  $j \leftarrow \text{index}(w_{k+1})$;
  if label$[k] = 1$ ***and*** **label**$[k+1] = 1$ **then**
    for $t \leftarrow i$ **to** j **do**
      $\mathbf{s}[t] \leftarrow 1$;

for $i \leftarrow 1$ **to** N **do**
  $\tilde{a}_i \leftarrow (a_i, \mathbf{s}[i])$;
$\tilde{\mathcal{A}} \leftarrow \langle \tilde{a}_1, \ldots, \tilde{a}_N \rangle$;

Output: Augmented sequence $\tilde{\mathcal{A}} = \langle (a_1, s_1), \ldots, (a_N, s_N) \rangle$

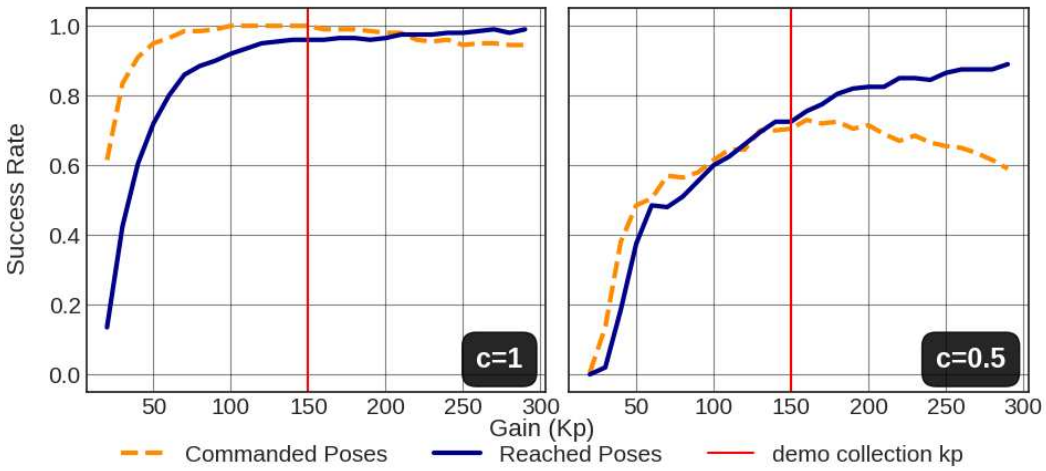


Figure G.4: **Demo replay at different speeds and controller gains.** We examine the effects of increasing controller gains and speed for replaying demos in simulation. Left: using commanded poses performs better when replaying at the original speed ($c = 1$) but using reached poses matches performance when using high gains. Right: A high-gain controller using reached poses performs better than one using commanded poses at a higher execution speed.

noisier reference trajectories. **High-gain controllers are more sensitive to noisy reference trajectories.** As seen in Fig. G.5, 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.

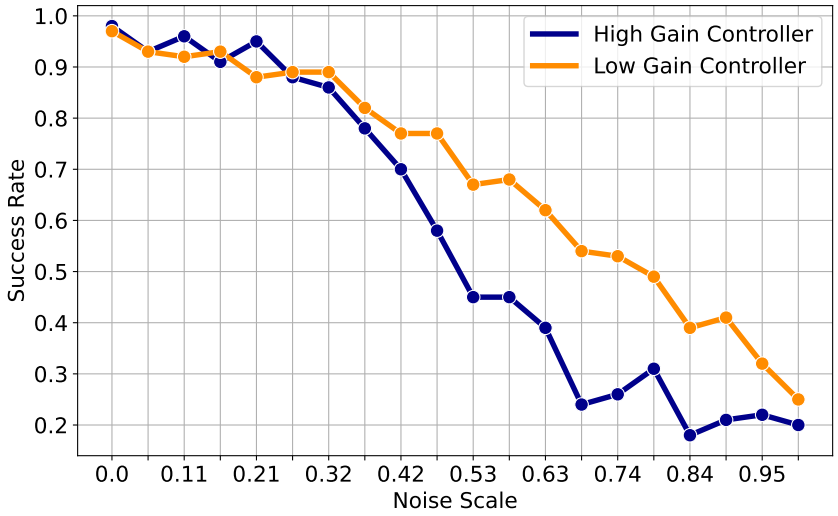


Figure G.5: **Noise vs high-gain and low-gain controller.** We study the effects of increasingly noisy actions with a high-gain and low-gain controller. Success Rate is averaged over 100 rollouts per noise scale. At higher noise levels, the success rate when using a high-gain controller drops more significantly than with a low-gain controller.

G.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 App. B for details). We assess these metrics across five tasks at a speed-up factor of 0.2. As delineated in Table G.1, 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.

H EAG Experiments

We first describe Error Adaptive Guidance in Alg. 2.

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

- ***H-EAG1***: Consistency-guiding is beneficial when action condition is aligned with observation
- ***H-EAG2***: Policy speed-up results in more misalignment between observation and action condition
- ***H-EAG3***: Tracking error is highly correlated to observation-action misalignment

Table G.1: **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 [8]	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 [8]	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 [8]	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 [8]	**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 [8]	0.40	**-2.62**	0.831	**0.679**
	Baseline	0.59	-2.88	0.276	1.210

Algorithm 2: Error Adaptive Guidance for Diffusion Policy

Input: Diffusion Policy π, observations o, future actions a^{c}, current end-effector pose $(\mathbf{x}_{\mathrm{pos}}, \mathbf{x}_{\mathrm{ori}})$, desired pose $(\mathbf{a}_{\mathrm{pos}}, \mathbf{a}_{\mathrm{ori}})$, error thresholds (pos_{teb}, ori_{teb})

▷ Calculate tracking error

$\Delta\mathbf{p} = \mathbf{a}_{\mathrm{pos}} - \mathbf{x}_{\mathrm{pos}}$;

$e_{\mathrm{pos}} = \|\Delta\mathbf{p}\|$;

$R_{\triangle} = R(\mathbf{a}_{\mathrm{ori}})^{\top} R(\mathbf{x}_{\mathrm{ori}})$;

$e_{\mathrm{ori}} = \arccos\Big(\frac{\mathrm{tr}(R_{\triangle}) - 1}{2}\Big)$;

▷ Add action conditioning to observations

if $e_{\mathrm{pos}} > pos_{teb}$ ***or*** $e_{\mathrm{ori}} > ori_{teb}$ **then** $o \leftarrow o \cup \emptyset$;

else $o \leftarrow o \cup a^{\mathrm{c}}$;

▷ Generate next set of actions

$\mathcal{A} \leftarrow \pi(o)$;

Output: Action sequence $\mathcal{A} = \langle a_1, a_2, \ldots, a_N \rangle$

H.1 Testing *H-EAG1*: 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-EAG1***: 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 Fig. H.6.

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.

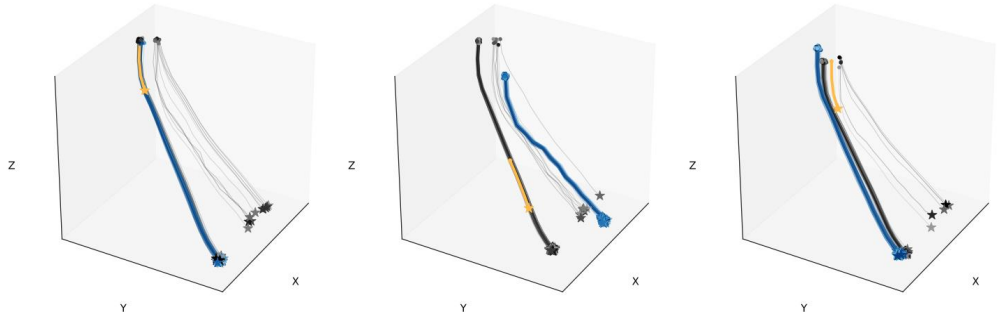


Figure H.6: **Impact of Action Conditioning on Consistency Guidance.** We evaluate how our algorithm adheres to conditioning on predicted actions. The action conditioning is shown in yellow; samples from the unconditional action distribution are shown in grey; and the conditional action distribution 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.

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.

H.2 Testing *H-EAG2*: 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 Sec. 4.1, 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 App. H.1, 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)** [46]: 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** [47]: We quantify how close the action condition is to its nearest neighbors 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)** [48]: 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 Fig. H.7. 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.

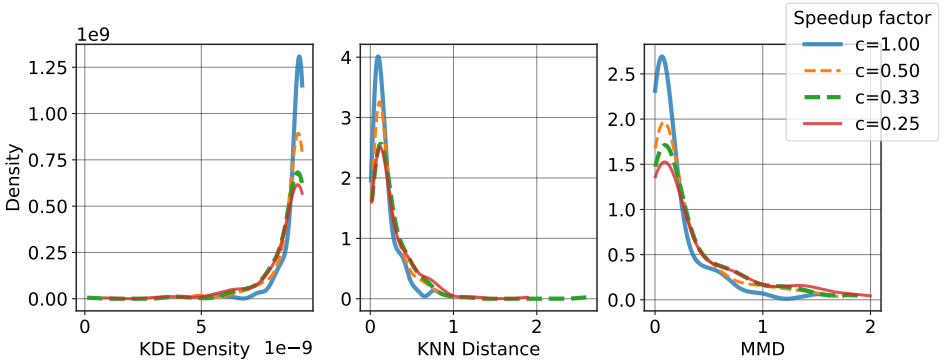


Figure H.7: **Effect of Policy Speed-up on Out-of-Distribution Action Conditions.** This figure evaluates how increasing the policy speed-up factor leads to mismatches between the action condition and the unconditional action distribution. We assess this by computing how well action conditions derived 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.

H.3 Testing *H-EAG3*: 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 App. H.2. However, these methods are computationally expensive and impractical for real-time deployment. Instead, we propose tracking error as a computationally efficient alternative.

Algorithm 3: Aggregate Actions (baseline)

Input: Sequence of delta Cartesian actions $A = \langle a_1, \ldots, a_T \rangle$

$AggActions \leftarrow []$
$curr \leftarrow a_1$
for $a \in A[2:]$ **do**
    if $\|curr\| > 0.05$ *cm* ***or*** $\text{dot}(a, curr) < 0.25$ **then**
        append *curr* to *AggActions*
        $curr \leftarrow a$
    else
        $curr \leftarrow curr + a$
append *curr* to *AggActions*

Output: Aggregated action sequence *AggActions*

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:

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

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 $\text{tr}(\cdot)$ 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 App. H.2. Specifically, we analyze the correlation separately for position and orientation tracking errors.

Fig. H.8 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. Fig. H.9 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.

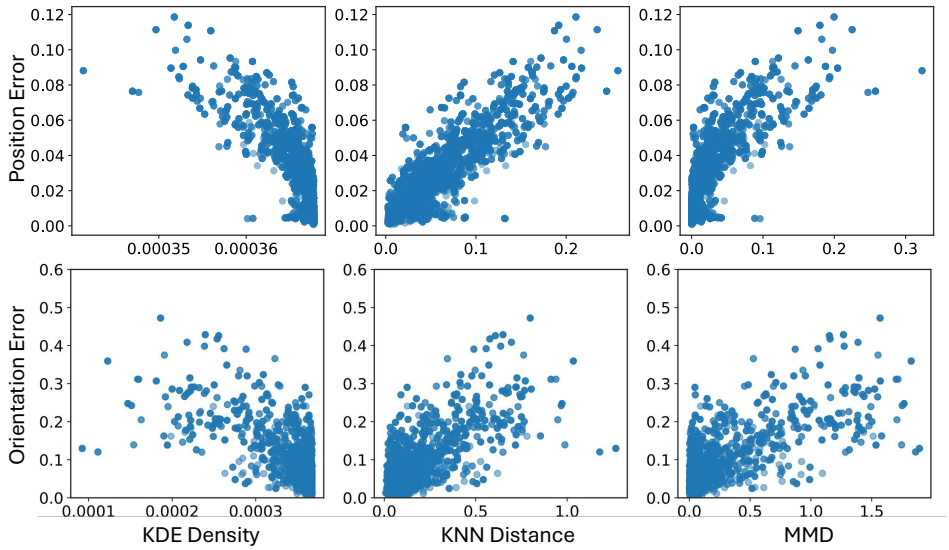


Figure H.8: **Correlation between Tracking Error and Out-of-Distribution Action Condition.** This figure illustrates the relationship between the tracking error and out-of-distribution (OOD) scores of action-conditions, computed using KDE Density (left), KNN distance (center), and MMD (right). The top 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.

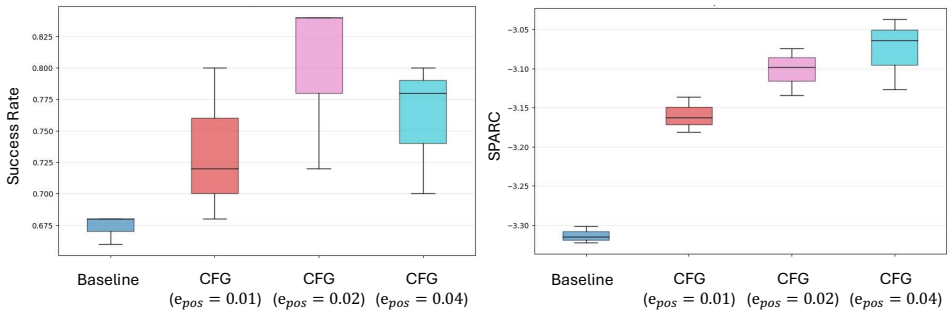


Figure H.9: **Effect of Tracking Error Threshold on Policy Performance.** This figure shows how different tracking error thresholds influence policy performance, measured by success rate (right) and trajectory smoothness (left). Each method is evaluated over 50 scenarios, repeated across three trials. 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.

Table I.2: Key Parameters of Policy Architecture for Simulation Experiments

Parameter	Value
General Settings	
Algorithm	diffusion_policy
Sequence Length	32
Frame Stack	2
Batch Size	128
Num. Epochs	1000
Horizon Settings	
Observation Horizon	2
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
CFG Conditioning	
Conditioning Horizon (H^{f})	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 (76×76, 1 crop)

I Hyperparameters and Baseline Implementation

The hyperparameters for our Policy backbone are listed in Table I.2. The parameters for the consistency guiding is listed in Table I.3 and the controller parameters in simulation are listed in Table I.4. The controller parameters for the real robot are listed in Table I.5. Additionally, we describe the algorithm for aggregating actions, which is one of our baselines, in Alg. 3.

Table I.3: Optimal Hyperparameter for Error Adaptive Guidance

Task	CFG weight	TEB (Ori.)	TEB (Pos.)
Can	1	0.05	0.02
Lift	2	0.05	0.02
Square	2	0.05	0.02
Mug Cleanup	2	0.05	0.02
Stack	1	0.03	0.01

J Detailed Ablation and SAIL with ACT

A more detailed ablation of our method is provided in Table J.6. The ablations are SAIL without the high gain controller **-HG**, without adaptive slowdown **-AS**, without Error Adaptive Guidance

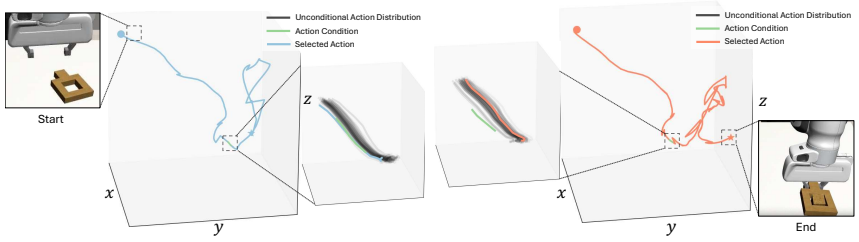
Table I.4: Controller and SAIL parameters for each simulated task

	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 I.5: 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

-EAG, and using **Commanded Poses** instead of reached poses. Additionally, we include a version of **DP-Fast** with a high-gain controller, listed as **DP-FHG**. Furthermore, we present results of SAIL with ACT [2] in Table J.7, with the only difference being the use of temporal ensembling as opposed to EAG for maintaining consistency between consecutive predictions.



(a) Successful rollout trajectory with action-conditioning.

(b) Successful rollout trajectory without action-conditioning.

Figure I.10: **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 (Fig. I.10a, blue trajectory) and without action-conditioning (Fig. I.10b, 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 J.6: Results of Ablation in Sim

		DP [1]	DP-FHG	SAIL	-HG	-AS	-EAG	Commanded Poses
Lift	SR ↑	**1.00**	0.53	**1.00**	0.67	0.97	0.98	0.89
	TPR↑	0.46	0.38	**1.68**	0.25	1.57	1.58	1.8
	ATR↓	2.23	1.28	**0.61**	3.39	0.63	0.63	0.52
	SOD↑	1.08	1.89	**3.98**	0.71	3.85	3.84	4.63
Can	SR↑	**0.97**	0.56	0.92	0.83	0.95	0.89	0.63
	TPR↑	0.18	0.39	0.51	0.18	**0.60**	0.50	0.37
	ATR↓	5.52	**1.28**	1.81	4.36	1.61	1.79	1.65
	SOD↑	1.05	**4.54**	3.20	1.33	3.60	3.23	3.52
Square	SR↑	0.83	0.18	**0.86**	0.59	0.64	0.79	0.31
	TPR↑	0.10	0.05	0.13	0.06	**0.25**	0.13	0.04
	ATR↓	7.56	2.54	6.41	8.8	**2.50**	5.78	4.25
	SOD↑	0.99	2.96	1.18	0.86	**3.01**	1.31	1.77
Stack	SR↑	**1.00**	0.80	0.98	0.90	0.94	0.95	0.82
	TPR↑	0.19	0.64	**0.66**	0.15	0.61	0.62	0.43
	ATR↓	5.50	**1.33**	1.56	6.6	1.71	1.56	2.81
	SOD↑	0.98	**4.08**	3.47	0.86	3.15	3.46	1.92
Mug	SR↑	0.68	0.54	**0.72**	0.53	0.44	0.68	0.54
	TPR↑	0.03	0.02	**0.08**	0.01	0.07	**0.08**	0.03
	ATR↓	17.44	19.18	8.09	18.24	**5.37**	8.15	17.38
	SOD↑	0.97	0.88	2.09	0.92	**3.14**	2.07	0.92

Table J.7: ACT experiments with and without SAIL

	Lift				Can				Square				Stack			
	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑	SR↑	TPR↑	ATR↓	SOD↑
ACT	**1.00**	0.40	2.69	0.90	**0.77**	0.12	4.04	0.96	**0.51**	0.04	8.21	0.92	**0.73**	0.11	6.18	0.87
SAIL	0.94	**1.50**	**0.65**	**3.71**	0.56	**0.25**	**2.12**	**2.72**	0.43	**0.11**	**3.22**	**2.34**	0.68	**0.28**	**3.14**	**1.72**