

# Simulation for Evaluating Robot Policies Should Be *Asynchronous* and *Real-Time*

Website: <https://anonymous.4open.science/w/async-robosim>

Anonymous Author(s)  
Affiliation  
Address  
email

1     **Abstract:** Simulation is an indispensable tool for evaluating robot algorithms in a  
2     safe and scalable manner. However, current simulation benchmarks introduce an  
3     artificial sim-to-real gap by running robot policies *synchronously* *i.e.* pausing the  
4     simulation to let the policy “think” for as long as it needs before executing actions.  
5     But real world physics does not wait for policy inference. Synchronous evaluation  
6     provides an unfair advantage to heavyweight policies with large backbones or  
7     iterative denoising that may be performant, but are too slow in the real world. We  
8     introduce AsyncRoboSim: a protocol to make any robotic simulation benchmark  
9     asynchronous by running the simulator and policy in parallel threads. The simulator  
10    runs in real time and communicates with the policy via shared inter-process queues  
11    for observations and actions. Importantly, the simulation rate is decoupled from  
12    the speed at which the policy consumes observations and produces actions. We  
13    instantiate our approach in `robosuite` [2], the underlying simulation framework  
14    for many robotic benchmarks [3–6] that can automatically inherit our asynchronous  
15    implementation. We show that the performance of standard manipulation tasks in  
16    AsyncRoboSim is significantly more correlated with real-world performance than  
17    conventional synchronous simulation. Realtime, asynchronous simulation naturally  
18    reflects inference latency in task performance metrics and also aids debugging.

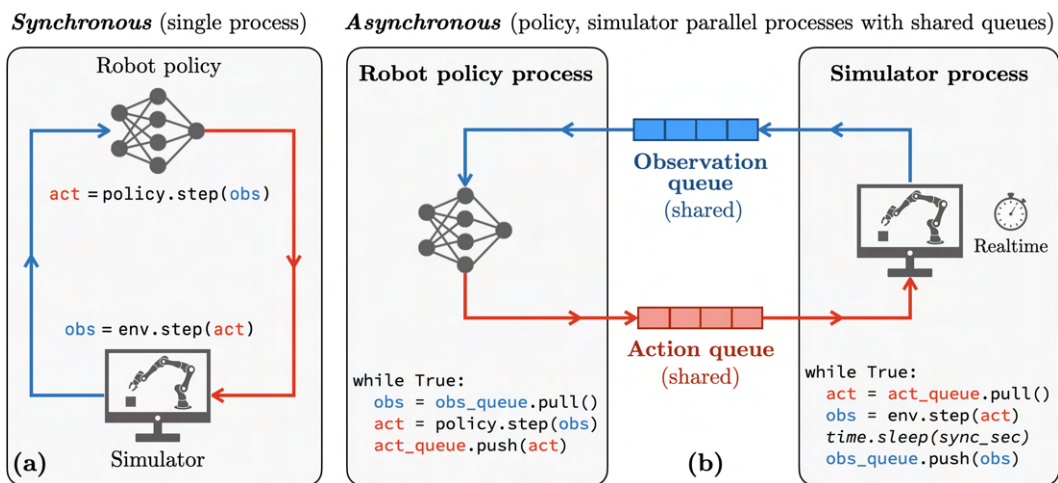


Figure 1: **(a)** Conventional *synchronous* simulation alternates between a policy inference step and a simulation step in a single process where the simulator is frozen during policy inference. **(b)** *Asynchronous* simulation runs the policy and simulator in separate parallel processes that communicate via shared observation and action queues. The simulator pulls actions and produces observations at fixed rates. Importantly, it synchronizes physics to match the *real-time* wall clock. The policy may pull observations and push actions at any rate it chooses. By decoupling the policy and simulator in parallel processes and communicating asynchronously via shared queues, the policy latency does not affect simulation. This allows us to fairly evaluate the affect of inference latency on policy performance, similar to how latency affects the policy’s behavior in the real-world.

## 19 **1 Introduction and related work**

20 Robotic simulation [7–13] is a critical tool used for prototyping, debugging, and importantly, eval-  
21 uating robot policies in a scalable manner while avoiding expensive real-world experiments. The  
22 dominant simulation paradigm used for robotic evaluation is largely inherited from the reinforcement  
23 learning community, particularly the OpenAI Gym API [1]. This API alternates between single  
24 steps of simulation and policy inference, namely (1) `obs = env.step(action)` and (2) `action =`  
25 `policy.step(obs)`. Notably, the simulator is paused when the policy inference step is executed.  
26 The overwhelming majority of robotic simulation benchmarks (in fact, every public benchmark we  
27 are aware of) [2–6, 12–18] conform to this synchronous API. The real world, however, does not  
28 wait for policy inference. Synchronous simulation provides an unfair advantage to slow policies  
29 with large VLM backbones or multiple diffusion steps since they are allowed to take an arbitrarily  
30 long time to generate the next action(s) while simulation remains paused. However, high latencies  
31 can be detrimental to policy performance in the real-world. Synchronous simulation produces an  
32 artificial sim-to-real gap that we call the “sync-to-async” gap. We argue that *asynchronous simulation*  
33 is necessary for accurately benchmarking robot policies and VLAs in the face of inference latencies.

34 ***Parallel simulator and policy inference:*** We introduce `AsyncRoboSim`: a specification to convert  
35 any simulation benchmark based on the synchronous API [1] into an asynchronous one by running  
36 the simulation and policy inference in separate processes that communicate via inter-process commu-  
37 nication (IPC). Specifically, we construct two shared queues: a *control queue* and an *observation*  
38 *queue*. The simulator process pulls actions from the control queue at a fixed control rate (say 500Hz)  
39 and pushes observations to the observation queue at a fixed camera rate (say 30Hz). The policy  
40 process is free to send actions and receive observations at any rate, that generally depends on its  
41 inference latency. If the policy fails to update the control queue in time for the next control step, the  
42 simulator uses the most recent action. The asynchronous design decouples the speed of the policy  
43 from the simulation rate, which is run in realtime. We instantiate our approach in `robosuite` [2]: a  
44 MuJoCo-based [7] simulation framework that many [3–6] robotic benchmarks build upon. Therefore,  
45 our asynchronous implementation can automatically be inherited by these benchmarks [3–6]. To  
46 the best of our knowledge, `AsyncRoboSim` is the *first public asynchronous and realtime simulation*  
47 *meta-benchmark for evaluating robot policies*. `AsyncRoboSim` can be used to evaluate both imitation  
48 learning and reinforcement learning policies.

49 ***What about real-world benchmarks?*** In contrast to simulation, real-world benchmarks like  
50 RoboArena [19], AutoEval [20] and ManipulationNet [21] directly evaluate robot policies on real  
51 robots. While they are inherently asynchronous and do not suffer from the “sync-to-async” gap,  
52 real-world benchmarking is cumbersome as robots can be expensive to maintain. Furthermore,  
53 real-world experiments are challenging to scale and hard to reproduce. On the flip side, simulation  
54 benchmarks can be scaled to a large diversity of scenes and objects, are reproducible, and can  
55 be run from the comfort of one’s personal computer without needing access to physical robots.  
56 Indeed, simulation and benchmarking have enabled tremendous progress in computer vision [22]  
57 and reinforcement learning [15, 23–26]. Prior works on simulation benchmarks focus on reducing  
58 well-known “sim-to-real gap” in photorealism of image rendering [18, 27, 28], sensor simulation and  
59 physics accuracy [9]. This progress is orthogonal to our work. We posit that simulation benchmarks  
60 need not suffer from yet another, artificially introduced “sync-to-async” gap: the simulator and policy  
61 can and should be run asynchronously. *Asynchronous simulation benchmarking* provides an important  
62 middle ground: it allows us to evaluate the speed-accuracy tradeoff of robot policies that is critical  
63 to real-world deployment, while being accessible, scalable and reproducible. In [Sec. 3](#), we show  
64 that the performance in asynchronous simulation is significantly more correlated with real-world  
65 performance of the same task (see [Fig. 2](#)), compared to the performance predicted by conventional  
66 synchronous simulation.

67 ***Unifies latency and performance metrics:*** Synchronous simulation requires policy “performance”  
68 (e.g. success rate or task completion time) and “latency” be reported as distinct metrics. However,  
69 latency is only as important as the extent to which it affects performance. For example, latency  
70 can affect dynamic tasks more adversely than quasi-static tasks. The advantage of asynchronous

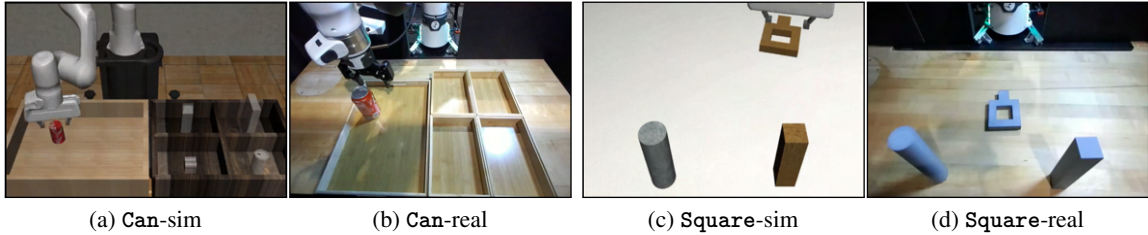


Figure 2: We reproduce the **Can** (a, b) and **Square** (c, d) tasks from the Robomimic [3] simulation benchmark (a, c) in the real-world (b, d). We find that policy performance measured by asynchronous simulation is more correlated with real-world performance than conventional synchronous simulation. See Sec. 3 for details.

71 simulation is that it naturally reflects the policy’s latency in downstream performance, thereby  
 72 unifying the two in a single metric. Asynchronous simulation is particularly well-suited to evaluate  
 73 methods that are designed to reduce latency and improve smoothness of policies, such as Real-Time  
 74 Chunking [29, 30], VLASH [31] and streaming policies [32]. In Sec. 3, we find that the higher the  
 75 latency of a policy, the worse is the performance estimate of conventional synchronous simulation.

76 **Useful for debugging:** We found that asynchronous simulation is an extremely useful debugging  
 77 tool that clearly visualizes the latency of robot policies. For example, it is common for diffusion  
 78 policies [33] to “jitter” at the boundaries of action chunks where the diffusion inference takes place.  
 79 This jittery motion profile was clearly visible in AsyncRoboSim matching real-world behavior. This  
 80 allows users to debug and fix latency-related issues in their robot policies even before deploying them  
 81 in the real world. In contrast, the motion in synchronous simulation appears deceptively smooth.

82 **Realtime rate control and monitoring:** Our implementation supports a controllable “realtime-rate”  
 83 (RTR) *i.e.* the speed of simulation compared to wall-clock time. We default the realtime rate to 1.  
 84 Modifying the RTR allows the user to evaluate policies under different hardware conditions and  
 85 resource constraints by speeding up or slowing down physics (equivalently, slowing down or speeding  
 86 up policy inference). The former simulates reduced compute resources *e.g.* on edge devices or  
 87 resource contention with other processes. The latter simulates scenarios where compute resources  
 88 might improve *e.g.* advances in GPU/microprocessor technology. Furthermore, the user’s hardware  
 89 might be too slow to support a desired RTR. In that case, AsyncRoboSim supports a *realtime rate*  
 90 *monitor* that measures the realized RTR and reports an error when it falls below the desired threshold.  
 91 Such monitoring prevents reporting inaccurate evaluation metrics when the system’s hardware cannot  
 92 support asynchronous simulation at the target realtime rate. In practice, we found all hardware  
 93 platforms we tested with were able to support RTR of at least 3 when using asynchronous MuJoCo  
 94 simulation. In Sec. 3, we find that increasing RTR (*i.e.* faster simulation) reduces task success rate  
 95 *and* widens the performance gap between synchronous and asynchronous simulation.

## 96 2 Method

97 **Parallel architecture and instantiation in *robosuïte*:** As shown in Fig. 1, the AsyncRoboSim  
 98 architecture is organized into two parallel processes: one that runs the policy and the other that  
 99 runs the simulator. We implement our approach in *robosuïte* [2]: a MuJoCo-based [7] simulation  
 100 framework that many [3–6] robotic benchmarks build upon. We implement our approach using  
 101 Python’s `multiprocessing` [34] library, and interface with the MuJoCo physics engine [7] via the  
 102 `mujoco` Python library. The policy and simulator processes communicate via shared observation and  
 103 action queues that are thread-safe and multiprocessing-safe. This architecture decouples the speed of  
 104 the policy from the speed of the simulator. Since neither process waits on each other, the simulator is  
 105 not throttled by slow policies. Each process runs in a loop until a shutdown signal is received.

106 **Policy process loop:** The policy process is free to implement any multiprocessing logic; we however  
 107 provide a default logic for converting any synchronous policy implementation to an asynchronous

108 one. This implementation continuously fetches the last observations from the observation queue,  
109 performs inference to predict the next action or action-chunk, and pushes the actions to the action  
110 queue.

111 **Simulator process loop:** The simulator process is responsible for (1) stepping through the physics  
112 engine, (2) consuming actions and emitting observations, and (3) maintaining a desired real-time rate  
113 (RTR) that defaults to 1. We define a configurable camera rate (defaulting to 30Hz) and a control  
114 rate (defaulting to 500Hz). `AsyncRoboSim` emits control events and observation events at these rates.  
115 The simulator main loop is implemented [here](#)<sup>1</sup>. At the beginning of the main loop, the simulator  
116 pulls the last action from the action queue. If no action is available (for *e.g.* due to a slow policy) the  
117 simulator uses the previously executed action, mirroring how a real robot continues actuating under  
118 its last command. It then performs one physics step via the synchronous `step(action)` API call  
119 implemented by most simulators. The physics step in MuJoCo defaults to 2ms of simulated time.  
120 At every control or observation event, the simulator synchronizes time with the wall-clock to run  
121 at the desired real-time rate (see paragraph below). It then computes observations (e.g. rendering  
122 camera images) and rewards and pushes them to the observation queue. It also periodically updates  
123 the interactive viewer and monitors for the end of the episode or shutdown signals. All actions and  
124 observations are timestamped in simulation time.

125 **Real-time rate synchronization:** The simulator periodically synchronizes with the wall-clock to  
126 ensure that the simulation time progresses at the desired real-time rate. At every control event  
127 (default: 500Hz) or an observation event (dictated by camera frequency), the simulator sleeps via a  
128 Python `time.sleep()` call to eliminate the time drift between the simulation and wall-clock time.  
129 Concretely, let  $\Delta_{\text{sim}}$ ,  $\Delta_{\text{real}}$  be the elapsed simulation time and real (wall-clock) time, respectively,  
130 since the last synchronization event. Given  $r > 0$ , simulation is considered perfectly synchronized  
131 with wall-clock time at real-time rate  $r$  if  $\Delta_{\text{sim}} = r \cdot \Delta_{\text{real}}$ . For example, if  $r = 2$ , one second of wall-  
132 clock time should correspond to two seconds of simulated time. However, most simulators including  
133 MuJoCo usually run significantly faster than real-time rate, *i.e.*  $\Delta_{\text{sim}} \gg r \cdot \Delta_{\text{real}}$ . Therefore, at every  
134 synchronization event, the simulator sleeps for  $\Delta_{\text{sleep}} = \max(0, \Delta_{\text{sim}}/r - \Delta_{\text{real}})$  seconds via a  
135 Python `time.sleep()` call; this logic is implemented [here](#)<sup>2</sup>. We are able to guarantee that simulated  
136 time never drifts from wall-clock (real) time by more than  $1/\text{control\_frequency}$  (default: 2ms).  
137 Since  $\Delta_{\text{sim}}$  and  $\Delta_{\text{real}}$  are computed relative to previous synchronization events, the pacing remains  
138 accurate over long horizons without accumulating drift. The real-time rate is a tunable knob that  
139 when set to  $r > 1$  emulates scarcer compute on edge devices or on heavy-load workstations, whereas  
140  $r < 1$  emulates fast compute due to hardware advances. The configurable real-time rate allows  
141 `AsyncRoboSim` to evaluate policies on hypothetical compute capabilities, although we default it to 1.

142 **Real-time rate monitoring:** It is conceivable that the user might be running asynchronous simulation  
143 on hardware with limited compute where  $\Delta_{\text{sim}} < r \cdot \Delta_{\text{real}}$  *i.e.* the simulation is running behind the  
144 required real-time rate. We *monitor* the running real-time rate and raise an error if the hardware is  
145 too slow to support the desired real-time rate, ensuring that real-time rates are guaranteed to the user.

### 146 3 Experiments

147 We structure our experimental evaluation around three central research questions: **(RQ1) Real-world**  
148 **correlation:** *does asynchronous simulation better predict real-world performance?* **(RQ2) Latency-**  
149 **driven gap:** *how does latency affect the “sync-to-async” gap?* **(RQ3) Realtime-rate sensitivity:** *how*  
150 *does the real-time rate affect the severity of synchronous overestimation?* To answer these questions,  
151 we perform experiments in Robomimic [3] and LIBERO [4], both of which extend `robosuite` [2].

152 **Experimental setup:** For Robomimic, we evaluate each policy on a desktop workstation equipped  
153 with an NVIDIA RTX 4060 GPU. Each task is evaluated with 400 trials per policy under both  
154 synchronous and asynchronous settings. We report success rates with 95% Wilson score confidence

<sup>1</sup>[https://anonymous.4open.science/r/realtime-robosuite/robosuite/environments/async\\_env.py#L343](https://anonymous.4open.science/r/realtime-robosuite/robosuite/environments/async_env.py#L343)

<sup>2</sup>[https://anonymous.4open.science/r/realtime-robosuite/robosuite/environments/async\\_env.py#L315-L323](https://anonymous.4open.science/r/realtime-robosuite/robosuite/environments/async_env.py#L315-L323)

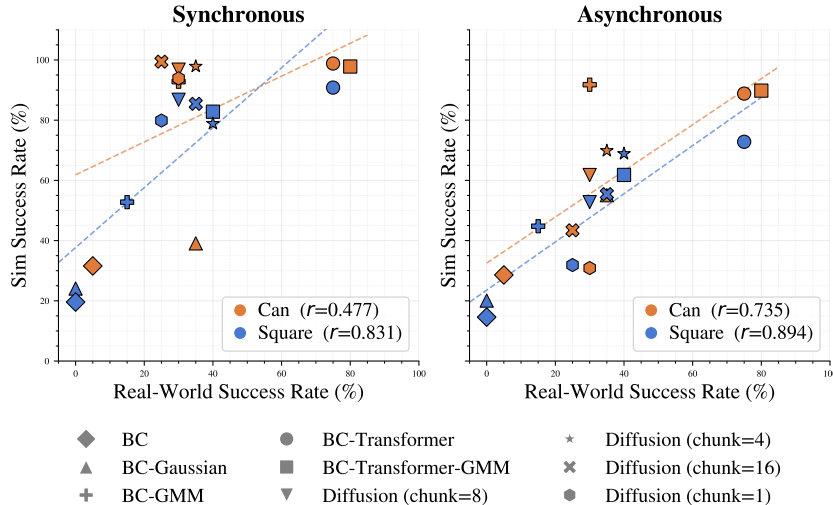


Figure 3: Linear correlation between simulation and real-world success rates across all evaluated policies under synchronous and asynchronous conditions.

Task	Condition	Pearson $r$	Spearman $\rho$	Pairwise Acc.
Can	Synchronous sim performance	0.477	0.485	0.724
	Asynchronous sim performance	0.735	0.664	0.812
	Improvement (sync $\rightarrow$ async)	<b>+0.258</b>	<b>+0.179</b>	<b>+0.088</b>
Square	Synchronous sim performance	0.831	0.782	0.824
	Asynchronous sim performance	0.894	0.975	0.971
	Improvement (sync $\rightarrow$ async)	<b>+0.063</b>	<b>+0.193</b>	<b>+0.147</b>

Table 1: Sim-to-real correlation metrics by task and sync/async simulation.

155 intervals [35]. For LIBERO, we evaluate each model on a workstation equipped with an NVIDIA  
 156 RTX 4090 GPU, using 100 trials per task suite following the standard LIBERO evaluation protocol.  
 157 For real-world evaluation, we use a Franka Research 3 robot on the **Can** and **Square** tasks. Each  
 158 policy uses multi-view RGB observations from a static front camera and a wrist-mounted camera,  
 159 resized to  $84 \times 84 \times 3$  images, and is evaluated over 20 trials. Training uses 63 demonstrations  
 160 for 200 training epochs on **Can** and 120 demonstrations for 400 training epochs on **Square**. Our  
 161 policy suite spans three policy families benchmarked in *robosuite* [2]: (1) MLP-based BC variants,  
 162 (2) Transformer-based BC variants, and (3) Diffusion Policy. The BC variants cover deterministic,  
 163 Gaussian, GMM, VAE, and transformer/transformer-GMM architectures. For diffusion policy, we  
 164 train the model to predict 16 action steps and evaluate execution chunk sizes  $T_a \in \{1, 2, 4, 8, 16\}$   
 165 by using only the first  $T_a$  predicted actions under both synchronous and asynchronous settings. The  
 166 spread of action chunk size provides a wide variation over policy performance and latencies.

167 **Real-world setup:** For real-world evaluation, we use the **Can** and **Square** tasks which are commonly  
 168 studied manipulation tasks from Robomimic. **Can** requires robust object transport across large spatial  
 169 and height variations, while **Square** is a contact-rich precision insertion task requiring tight geometric  
 170 alignment between a square nut and a peg. We reproduce the **Can** and **Square** in the real-world via  
 171 the same objects and 3D printing the same tools; see Fig. 2 for a visual comparison. This allows us to  
 172 measure real-world policy performance on the **Can** and **Square** tasks, as well as the performance in  
 173 their “digital twin” simulation (synchronous and asynchronous simulation). While these tasks are not  
 174 particularly dynamic, we find that they still exhibit a significant “sync-to-async” gap.

175 **Real-world results (RQ1):** We evaluate whether asynchronous simulation more faithfully reflects  
 176 real-world policy performance compared to synchronous simulation of the same. Fig. 3 and Table 1  
 177 compare policy success rates under synchronous simulation, asynchronous simulation, and real-world  
 178 deployment. We want to measure the correlation between simulation success rate and real-world

179 success rate, and compare whether this correlation is stronger for asynchronous simulation than  
 180 synchronous simulation. We measure sim-to-real correlation using three metrics:

181 1. **Pearson linear correlation ( $r$ ):** Given paired simulated and real-world success rates  $\{(x_i, y_i)\}_{i=1}^n$   
 182 across policies, with means  $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$  and  $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$ , the Pearson linear correlation is

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

183 This metric measures *linear* correlation, and is sensitive to the magnitude of differences.

184 2. **Spearman rank-correlation ( $\rho$ ):** Let  $k_i^{\text{real}}$  be rank of the  $i$ -th policy according to real-world  
 185 success, and  $k_i^{\text{sim}}$  be its rank according to simulated success. The Spearman rank correlation is  
 186 simply the linear correlation between ordinal ranks:

$$\rho = 1 - \frac{6 \sum_{i=1}^n (k_i^{\text{real}} - k_i^{\text{sim}})^2}{n(n^2 - 1)}$$

187 This metric measures rank correlation. It is insensitive to the magnitude of performance difference.  
 188 Instead, it focuses on the relative ordering of policies.

189 3. **Pairwise ranking accuracy (PRA):** across each possible pairs of robot policies, we check whether  
 190 the binary ranking according to real-world success rate matches that of the simulated success  
 191 rate. PRA is the fraction of pairs where the binary ranking matches. Given paired simulated and  
 192 real-world success rates  $\{(x_i, y_i)\}_{i=1}^n$  across policies, PRA is given by

$$\text{PRA} = \binom{n}{2}^{-1} \sum_{i < j} \mathbb{I}[\text{sign}(x_i - x_j) = \text{sign}(y_i - y_j)]$$

193 where  $\mathbb{I}[\cdot]$  is the indicator function. Like the Spearman correlation, this metric also measures rank  
 194 correlation and is only sensitive to relative ordering of policies.

195 **Fig. 3** visualizes the linear correlation between simulated and real-world success rates. Each point  
 196 (denoted by various symbols) represents a single policy evaluation (orange for the **Can** task and blue  
 197 for the **Square** task). The x-axis plots real-world success rate (ground-truth) while the y-axis shows  
 198 simulated success. Asynchronous simulation shows stronger linear correlation between real-world  
 199 performance compared to synchronous simulation on both **Can** and **Square** tasks. This is reflected  
 200 by the increase in the squared Pearson linear correlation ( $r$ ) from 0.477 to 0.735 on **Can** and from  
 201 0.831 to 0.894 on **Square**. **Table 1** shows the Spearman rank-correlation ( $\rho$ ) and Pairwise ranking  
 202 accuracy (PRA) between simulated and real-world success rates. Across both tasks, and across  
 203 all three metrics, asynchronous simulation consistently shows stronger correlation with real-world  
 204 performance than synchronous simulation. This indicates that asynchronous simulation is a better  
 205 predictor of real-world success and is able to better predict when one policy might outperform another.

206 As a representative example, we consider the **Square** task. Under conventional synchronous simula-  
 207 tion, diffusion policy with chunk size 8 outperforms BC-Transformer-GMM, with success rates of  
 208 86% and 83%, respectively. However, this ordering is reversed under both asynchronous simulation  
 209 and real-world deployment. Note that diffusion policy has substantially higher inference latency than  
 210 BC-Transformer-GMM (587 ms compared with 12 ms). The rank reversal suggests that synchronous  
 211 evaluation can misrepresent deployment-relevant policy rankings. Detailed per-policy success rates  
 212 and latency profiles are provided in **App. C**. Both **Fig. 3** and **Table 1** are derived from this data.

213 **Robomimic latency results (RQ2):** We evaluate policy performance under synchronous and  
 214 asynchronous simulation across all five Robomimic tasks: **Lift**, **Can**, **Square**, **Transport**, and  
 215 **ToolHang**. In **Fig. 4** (left), we quantify the impact of the policy’s inference latency on asynchronous  
 216 execution by reporting the drop in success rates between synchronous and asynchronous settings. We  
 217 report the complete table of the experimental data in **App. A** that is used to plot **Fig. 4**.

218 We find that low-latency MLP/GMM policies are less sensitive to the transition from synchronous to  
 219 asynchronous execution than high-latency policies. These policies have inference latencies between

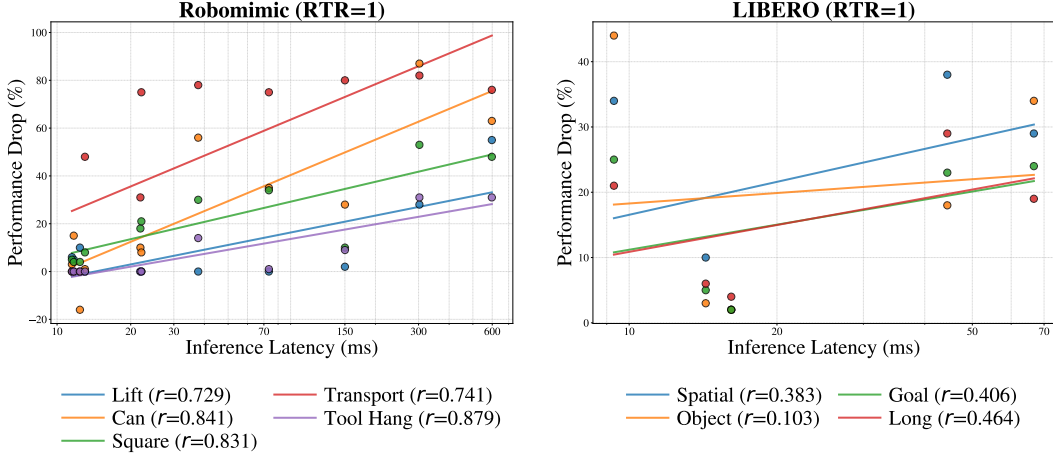


Figure 4: Performance drop (sync  $\rightarrow$  async, %) vs. inference latency across all evaluated policies on Robomimic (left) and LIBERO (right). Each point represents a single task-policy pair colored by task. Lines show per-task log-linear regression fits with slopes annotated in the legend.

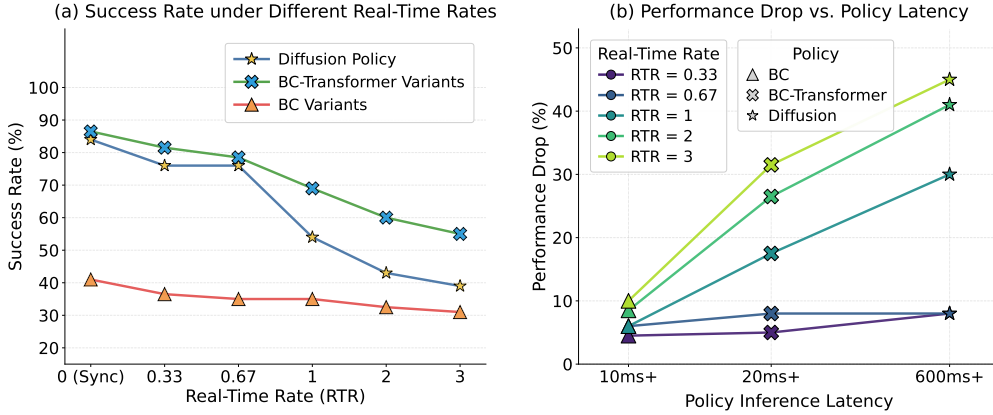


Figure 5: Analysis of success rate and “sync-to-async” gap on the Robomimic **Square** task across various robot policies. **Left:** Success rate reduces with increasing RTR (faster simulation), and this effect is more pronounced for higher-latency policies. **Right:** Performance drop from synchronous to asynchronous simulation increases with latency, and is more pronounced at higher RTRs. *Note:* synchronous simulation corresponds to RTR = 0.

220 11 ms and 22 ms, which are comparable to the simulator control-loop timescale. As a result, their  
 221 asynchronous performance largely preserves the trends observed under synchronous evaluation. For  
 222 example, low-latency policies achieve similar performance on the **Can** task under synchronous and  
 223 asynchronous evaluation. In contrast, diffusion policy [33] variants exhibit a substantially larger  
 224 gap between synchronous and asynchronous evaluation. Although these policies achieve strong  
 225 synchronous success rates, their iterative denoising process incurs much higher inference latency.  
 226 Under synchronous evaluation, this computation time is effectively hidden, as the simulator waits  
 227 for policy inference to complete. Under asynchronous evaluation, the simulator instead continues  
 228 to advance in real time executing the most recent available action, reducing closed-loop reactivity.  
 229 On the **Can** task, diffusion policy with chunk size 16 drops from 98% success under synchronous  
 230 evaluation to 42% under asynchronous evaluation.

231 **LIBERO latency results (RQ2):** We further study how synchronous simulation affects vision-  
 232 language-action (VLA) models, which are heavier robot policies with vision-language backbones. We  
 233 select a diverse set of VLA policies that span the spectrum of capability vs. latency. SmolVLA [36]  
 234 is a compact 450M-parameter model designed for efficient deployment, while X-VLA [37] and  
 235 MolmoAct2 [38] represent recent high-performing open VLA models with stronger action-reasoning  
 236 capabilities. We further include  $\pi_0$  [39] and  $\pi_{0.5}$  [40] as widely adopted generalist robot policies.

237 In Fig. 4 (right), we find that the correlation between inference latency and performance drop  
238 for VLAs on LIBERO is positive, but is weaker than the robot control policies in Robomimic.  
239 Asynchronous robustness may also depend on policy-level properties, such as temporal stability,  
240 action correction, and the ability to recover from stale commands. We find that X-VLA retains 93.8%  
241 of its performance and MolmoAct2 retains 97.6% of its performance going from synchronous to  
242 asynchronous simulation. In contrast,  $\pi_0$  retains only 68.6%. We observe self-correction behavior  
243 from MolmoAct2 in our asynchronous simulator. *Importantly, this behavior is not visible under*  
244 *synchronous simulation*, and is only visible when accounting for the asynchronous interaction between  
245 the policy and the environment. Full results on the LIBERO benchmark are provided in App. B.

246 **Real-time rate analysis (RQ3):** We now investigate whether the gap increases with real-time rate  
247 (RTR) *i.e.* when the simulator advances faster relative to policy inference. We vary RTR and measure  
248 how each policy’s success rate changes under different simulator speeds. Fig. 5 (left) shows the  
249 success rate as a function of RTR on the Robomimic Square task. Note that synchronous simulation  
250 is the limiting case of asynchronous simulation when  $RTR \rightarrow 0$  *i.e.* the simulation runs infinitely  
251 slowly and waits arbitrarily long for policy inference to complete. Increasing the real-time rate (RTR)  
252 amplifies the sync-to-async performance drop. Furthermore, this effect is strongly compounded by  
253 latency. Low-latency BC variants remain relatively stable as RTR increases, indicating that when  
254 policy inference takes only around 10 ms, the controller can still update actions fast enough to preserve  
255 closed-loop reactivity. In contrast, high-latency diffusion policy exhibits a steeper degradation. This  
256 reveals a key principle for robot evaluation: latency is not merely an auxiliary runtime statistic  
257 but directly affects task success when the asynchronous nature of the world is properly accounted  
258 for. Asynchronous simulation exposes a latency-reactivity tradeoff that synchronous evaluation  
259 systematically hides. Fig. 5 (right) shows that performance drop as a function of policy latency is  
260 exacerbated at high real-time rates *i.e.* when the asynchronous simulator is run faster.

## 261 4 Limitations

262 **Evaluation speed:** Since synchronous simulation need not adhere to real-time constraints, it can  
263 perform evaluations faster than realtime. Asynchronous simulation, however, is constrained to be  
264 slow with `time.sleep()` calls even if the hardware can support faster-than-realtime simulation.  
265 This is a price to pay for simulating latencies like they appear in the real-world. However, unlike  
266 real-world evals, asynchronous simulation can be run around-the-clock, unsupervised.

267 **Hardware constraints:** Asynchronous simulation may not be supported by some hardware (e.g.  
268 older computers, edge devices) that are too slow to attain real-time simulation speeds. Our real-time  
269 monitor can be used to detect such scenarios and report errors. That being said, modern simulators  
270 like MuJoCo are sufficiently fast and unlikely for most modern computers to be unable to support  
271 real-time rate. We found that all devices we tested could support at least 3x realtime speed.

## 272 5 Conclusion

273 We argue that simulation for evaluating robot policies should be *asynchronous* and *realtime*. We  
274 introduced AsyncRoboSim, a meta-benchmark that converts any conventional synchronous simula-  
275 tion benchmark into an asynchronous one by running the policy and simulator in parallel processes  
276 that communicate via shared observation and action queues, decoupling the simulation rate from the  
277 policy’s inference speed. We demonstrated that synchronous evaluation introduces an artificial “sync-  
278 to-async” gap that systematically overestimates the deployable performance of high-latency policies,  
279 and this gap grows with inference latency and the realtime rate. Crucially, asynchronous performance  
280 is more correlated with real-world performance and better preserves true policy rankings by incorpo-  
281 rating latency directly into task-performance metrics. We hope this work encourages the robotics  
282 community to adopt asynchronous, realtime simulation as a scalable, accessible, and reproducible  
283 middle ground between idealized synchronous simulation and costly real-world evaluation.

## References

- 284
- 285 [1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba.  
286 *Openai gym*. *arXiv preprint arXiv:1606.01540*, 2016.
- 287 [2] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, Y. Zhu, and K. Lin.  
288 *robosuite: a modular simulation framework and benchmark for robot learning*. In *arXiv*  
289 *preprint arXiv:2009.12293*, 2020.
- 290 [3] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese,  
291 Y. Zhu, and R. Martín-Martín. *robomimic: what matters in learning from offline human*  
292 *demonstrations for robot manipulation*. In *arXiv preprint arXiv:2108.03298*, 2021.
- 293 [4] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. *LIBERO: Benchmarking*  
294 *knowledge transfer for lifelong robot learning*. *Advances in Neural Information Processing*  
295 *Systems*, 36:44776–44791, 2023.
- 296 [5] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu.  
297 *Robocasa: Large-scale simulation of everyday tasks for generalist robots*. *arXiv preprint*  
298 *arXiv:2406.02523*, 2024.
- 299 [6] S. Nasiriany, S. Nasiriany, A. Maddukuri, and Y. Zhu. *Robocasa365: A large-scale simulation*  
300 *framework for training and benchmarking generalist robots*. *arXiv preprint arXiv:2603.04356*,  
301 2026.
- 302 [7] E. Todorov, T. Erez, and Y. Tassa. *MuJoCo: A physics engine for model-based control*. In *2012*  
303 *IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE,  
304 2012.
- 305 [8] E. Coumans and Y. Bai. *Pybullet, a python module for physics simulation for games, robotics*  
306 *and machine learning*, 2016.
- 307 [9] R. Tedrake et al. *Drake: Model-based design and verification for robotics*, 2019.
- 308 [10] G. Authors. *Genesis: A Generative and Universal Physics Engine for Robotics and Beyond*,  
309 December 2024. URL <https://github.com/Genesis-Embodied-AI/Genesis>.
- 310 [11] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin,  
311 A. Allshire, A. Handa, et al. *Isaac gym: High performance gpu-based physics simulation for*  
312 *robot learning*. *arXiv preprint arXiv:2108.10470*, 2021.
- 313 [12] S. Tao, F. Xiang, A. Shukla, Y. Qin, X. Hinrichsen, X. Yuan, C. Bao, X. Lin, Y. Liu, T.-k.  
314 Chan, et al. *Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable*  
315 *embodied ai*. *arXiv preprint arXiv:2410.00425*, 2024.
- 316 [13] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar,  
317 et al. *ORBIT: A unified simulation framework for interactive robot learning environments*.  
318 *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023.
- 319 [14] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. *RLBench: The robot learning benchmark &*  
320 *learning environment*. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.
- 321 [15] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. *Meta-World: A*  
322 *benchmark and evaluation for multi-task and meta reinforcement learning*. In *Conference on*  
323 *robot learning*, pages 1094–1100. PMLR, 2020.
- 324 [16] C. Li, R. Zhang, J. Wong, C. Gokmen, S. Srivastava, R. Martín-Martín, C. Wang, G. Levine,  
325 M. Lingelbach, J. Sun, et al. *BEHAVIOR-1K: A benchmark for embodied ai with 1,000*  
326 *everyday activities and realistic simulation*. In *Conference on Robot Learning*, pages 80–93.  
327 PMLR, 2023.

- 328 [17] C. Sferrazza, D.-M. Huang, X. Lin, Y. Lee, and P. Abbeel. [Humanoidbench: Simu-](#)  
329 [lated humanoid benchmark for whole-body locomotion and manipulation.](#) *arXiv preprint*  
330 *arXiv:2403.10506*, 2024.
- 331 [18] Y. Jangir, Y. Zhang, P.-C. Lo, K. Yamazaki, C. Zhang, K.-H. Tu, T.-W. Ke, L. Ke, Y. Bisk, and  
332 K. Fragkiadaki. [RobotArena  \$\infty\$ : Scalable Robot Benchmarking via Real-to-Sim Translation.](#)  
333 *arXiv preprint arXiv:2510.23571*, 2025.
- 334 [19] P. Atreya, K. Pertsch, T. Lee, M. J. Kim, A. Jain, A. Kuramshin, C. Eppner, C. Neary, E. Hu,  
335 F. Ramos, et al. [RoboArena: Distributed real-world evaluation of generalist robot policies.](#)  
336 *arXiv preprint arXiv:2506.18123*, 2025.
- 337 [20] Z. Zhou, P. Atreya, Y. L. Tan, K. Pertsch, and S. Levine. [AutoEval: Autonomous evaluation](#)  
338 [of generalist robot manipulation policies in the real world.](#) *arXiv preprint arXiv:2503.24278*,  
339 2025.
- 340 [21] Y. Chen, K. Kimble, E. H. Adelson, T. Asfour, P. Chanrungrameekul, S. Chitta, Y. Chitambar,  
341 Z. Chen, K. Goldberg, D. Kragic, et al. [ManipulationNet: An infrastructure for benchmark-](#)  
342 [ing real-world robot manipulation with physical skill challenges and embodied multimodal](#)  
343 [reasoning.](#) *arXiv preprint arXiv:2603.04363*, 2026.
- 344 [22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy,  
345 A. Khosla, M. Bernstein, et al. [Imagenet large scale visual recognition challenge.](#) *International*  
346 *journal of computer vision*, 115(3):211–252, 2015.
- 347 [23] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. [The arcade learning environment: An](#)  
348 [evaluation platform for general agents.](#) *Journal of artificial intelligence research*, 47:253–279,  
349 2013.
- 350 [24] Y. Duan, X. Chen, R. Houthoofd, J. Schulman, and P. Abbeel. [Benchmarking deep reinforcement](#)  
351 [learning for continuous control.](#) In *International conference on machine learning*, pages 1329–  
352 1338. PMLR, 2016.
- 353 [25] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki,  
354 J. Merel, A. Lefrancq, et al. [Deepmind control suite.](#) *arXiv preprint arXiv:1801.00690*, 2018.
- 355 [26] W. Zhao, J. P. Queralta, and T. Westerlund. [Sim-to-real transfer in deep reinforcement learning](#)  
356 [for robotics: a survey.](#) In *2020 IEEE symposium series on computational intelligence (SSCI)*,  
357 pages 737–744. IEEE, 2020.
- 358 [27] X. Li, K. Hsu, J. Gu, K. Pertsch, O. Mees, H. R. Walke, C. Fu, I. Lunawat, I. Sieh, S. Kir-  
359 mani, et al. [Evaluating real-world robot manipulation policies in simulation.](#) *arXiv preprint*  
360 *arXiv:2405.05941*, 2024.
- 361 [28] A. Jain, M. Zhang, K. Arora, W. Chen, M. Torne, M. Z. Irshad, S. Zakharov, Y. Wang, S. Levine,  
362 C. Finn, et al. [Polaris: Scalable real-to-sim evaluations for generalist robot policies.](#) *arXiv*  
363 *preprint arXiv:2512.16881*, 2025.
- 364 [29] K. Black, M. Galliker, and S. Levine. [Real-time execution of action chunking flow policies.](#)  
365 *Advances in Neural Information Processing Systems*, 38:33383–33407, 2026.
- 366 [30] K. Black, A. Z. Ren, M. Equi, and S. Levine. [Training-time action conditioning for efficient](#)  
367 [real-time chunking.](#) *arXiv preprint arXiv:2512.05964*, 2025.
- 368 [31] J. Tang, Y. Sun, Y. Zhao, S. Yang, Y. Lin, Z. Zhang, J. Hou, Y. Lu, Z. Liu, and S. Han. [VLASH:](#)  
369 [Real-time vlas via future-state-aware asynchronous inference.](#) *arXiv preprint arXiv:2512.01031*,  
370 2025.

- 371 [32] S. Jiang, X. Fang, N. Roy, T. Lozano-Pérez, L. P. Kaelbling, and S. Ancha. Streaming flow  
372 policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow  
373 trajectories. *arXiv preprint arXiv:2505.21851*, 2025.
- 374 [33] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion  
375 policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics*  
376 *Research*, 44(10-11):1684–1704, 2025.
- 377 [34] Python Software Foundation. multiprocessing — process-based parallelism. Python  
378 Standard Library documentation, 2024. URL [https://docs.python.org/3/library/  
379 multiprocessing.html](https://docs.python.org/3/library/multiprocessing.html).
- 380 [35] E. B. Wilson. Probable Inference, the Law of Succession, and Statistical Inference. *Journal of*  
381 *the American Statistical Association*, 22(158):209–212, 1927.
- 382 [36] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi,  
383 C. Pascal, M. Russi, A. Marafioti, et al. Smolvla: A vision-language-action model for affordable  
384 and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.
- 385 [37] J. Zheng, J. Li, Z. Wang, D. Liu, X. Kang, Y. Feng, Y. Zheng, J. Zou, Y. Chen, J. Zeng, et al.  
386 X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model.  
387 *arXiv preprint arXiv:2510.10274*, 2025.
- 388 [38] H. Fang, J. Duan, D. Clay, S. Wang, S. Liu, W. Huang, X. Fan, W.-C. Tsai, S. Chen, Y. R. Wang,  
389 et al. MolmoAct2: Action Reasoning Models for Real-world Deployment. *arXiv preprint*  
390 *arXiv:2605.02881*, 2026.
- 391 [39] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman,  
392 B. Ichter, et al.  $\pi_0$ : A Vision-Language-Action Flow Model for General Robot Control. *arXiv*  
393 *preprint arXiv:2410.24164*, 2024.
- 394 [40] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi,  
395 C. Finn, N. Fusai, et al.  $\pi_{0.5}$ : a Vision-Language-Action Model with Open-World Generalization.  
396 *arXiv preprint arXiv:2504.16054*, 2025.

# Appendix

## Simulation for Evaluating Robot Policies Should Be *Asynchronous* and *Real-Time*

Website: <https://anonymous.4open.science/w/async-robosim>

### 397 A Robomimic full results

398 We provide detailed per-policy results on the Robomimic benchmark [3] across all five tasks: **Lift**,  
 399 **Can**, **Square**, **Transport**, and **ToolHang** in Table 3. All values are success rates (%) evaluated  
 400 over 400 episodes under synchronous and asynchronous settings. 95% Wilson score confidence  
 401 intervals [35] are reported in brackets.

		Task Success Rate (%)						Perf. drop	Rel. drop (%)	Latency ↓
		Sync / Async								
		Lift	Can	Square	Trans.	T. Hang	Avg.			
1	BC	86 / 80	31 / 28	19 / 14	0 / 0	0 / 0	27.2 / 24.4	-2.8	10.29	11.5 ms
2	BC-VAE	82 / 77	87 / 72	28 / 24	0 / 0	0 / 0	39.4 / 34.6	-4.8	12.18	11.7 ms
3	BC-Gaussian	60 / 50	38 / 54	23 / 19	0 / 0	0 / 0	<b>24.2</b> / 24.6	+0.4	<b>-1.65</b>	12.4 ms
4	BC-GMM	100 / 100	94 / 93	54 / 46	52 / 4	0 / 0	60.0 / 48.6	-11.4	19.00	13.0 ms
5	BC-TF	100 / 100	98 / 88	90 / 72	33 / 2	0 / 0	64.2 / 52.4	-11.8	18.38	21.9 ms
6	BC-TF-GMM	100 / 100	98 / 90	83 / 62	78 / 3	0 / 0	71.8 / 51.0	-20.8	28.97	22.1 ms
7	DP (chunk=8)	100 / 100	96 / 61	86 / 52	81 / 6	30 / 29	78.6 / 49.6	-29.0	36.90	73.36 ms
8	DP (chunk=1)	100 / 45	95 / 32	81 / 33	76 / 0	31 / 0	76.6 / <b>22.0</b>	-54.6	<b>71.28</b>	597.80 ms
9	DP (chunk=4)	100 / 98	97 / 69	78 / 68	82 / 2	35 / 26	78.4 / <b>52.6</b>	-25.8	32.91	150.05 ms
10	DP (chunk=16)	100 / 100	98 / 42	84 / 54	83 / 5	46 / 32	<b>82.2</b> / 46.6	-35.6	43.31	37.73 ms
11	DP (chunk=2)	100 / 72	97 / 10	83 / 30	82 / 0	33 / 2	79.0 / 22.8	<b>-56.2</b>	71.14	302.40 ms

Table 2: Synchronous and asynchronous simulation performance on Robomimic [3].

Policy	Sync	Async	$\Delta$	Inference Latency
<b>Can</b>				
BC-TF	98 [96.1, 99.0]	88 [84.4, 90.8]	10	55.76 ms
BC-TF-GMM	98 [96.1, 99.0]	90 [86.7, 92.6]	8	56.01 ms
BC-Gaussian	38 [33.4, 42.8]	54 [49.1, 58.8]	-16	26.07 ms
BC	31 [26.7, 35.7]	28 [23.8, 32.6]	3	22.76 ms
BC-GMM	94 [91.2, 95.9]	93 [90.1, 95.1]	1	25.94 ms
BC-VAE	87 [83.3, 89.9]	72 [67.4, 76.2]	15	23.27 ms
DP (chunk=16)	98 [96.1, 99.0]	42 [37.3, 46.9]	56	574.39 ms
DP (chunk=8)	96 [93.6, 97.5]	61 [56.1, 65.7]	35	534.97 ms
DP (chunk=4)	97 [94.8, 98.3]	69 [64.3, 73.3]	28	602.67 ms
DP (chunk=2)	97 [94.8, 98.3]	10 [7.4, 13.3]	87	623.06 ms
DP (chunk=1)	95 [92.4, 96.7]	32 [27.6, 36.7]	63	587.53 ms
<b>Square</b>				
BC-TF	90 [86.7, 92.6]	72 [67.4, 76.2]	18	12.34 ms
BC-TF-GMM	83 [79.0, 86.4]	62 [57.2, 66.6]	21	12.38 ms
BC-Gaussian	23 [19.1, 27.4]	19 [15.5, 23.1]	4	7.13 ms
BC	19 [15.5, 23.1]	14 [10.9, 17.7]	5	6.72 ms
BC-GMM	54 [49.1, 58.8]	46 [41.2, 50.9]	8	7.75 ms

Continued on next page

Policy	Sync	Async	$\Delta$	Inference Latency
BC-VAE	28 [23.8, 32.6]	24 [20.1, 28.4]	4	6.79 ms
DP (chunk=16)	84 [80.1, 87.3]	54 [49.1, 58.8]	30	595.73 ms
DP (chunk=8)	86 [82.3, 89.1]	52 [47.1, 56.9]	34	587.12 ms
DP (chunk=4)	78 [73.7, 81.8]	68 [63.3, 72.4]	10	602.23 ms
DP (chunk=2)	83 [79.0, 86.4]	30 [25.7, 34.7]	53	589.04 ms
DP (chunk=1)	81 [76.9, 84.5]	33 [28.6, 37.8]	48	594.50 ms
<b>Lift</b>				
BC-TF	100 [99.0, 100.0]	100 [99.0, 100.0]	0	11.82 ms
BC-TF-GMM	100 [99.0, 100.0]	100 [99.0, 100.0]	0	11.76 ms
BC-Gaussian	60 [55.1, 64.7]	50 [45.1, 54.9]	10	7.21 ms
BC	86 [82.3, 89.1]	80 [75.8, 83.6]	6	6.85 ms
BC-GMM	100 [99.0, 100.0]	100 [99.0, 100.0]	0	7.86 ms
BC-VAE	82 [77.9, 85.5]	77 [72.6, 80.9]	5	6.96 ms
DP (chunk=16)	100 [99.0, 100.0]	100 [99.0, 100.0]	0	614.90 ms
DP (chunk=8)	100 [99.0, 100.0]	100 [99.0, 100.0]	0	604.83 ms
DP (chunk=4)	100 [99.0, 100.0]	98 [96.1, 99.0]	2	594.14 ms
DP (chunk=2)	100 [99.0, 100.0]	72 [67.4, 76.2]	28	602.35 ms
DP (chunk=1)	100 [99.0, 100.0]	45 [40.2, 49.9]	55	596.97 ms
<b>Transport</b>				
BC-TF	33 [28.6, 37.8]	2 [1.0, 3.9]	31	16.82 ms
BC-TF-GMM	78 [73.7, 81.8]	3 [1.7, 5.2]	75	17.78 ms
BC-Gaussian	0 [0.0, 1.0]	0 [0.0, 1.0]	0	13.48 ms
BC	0 [0.0, 1.0]	0 [0.0, 1.0]	0	13.12 ms
BC-GMM	52 [47.1, 56.9]	4 [2.5, 6.4]	48	14.19 ms
BC-VAE	0 [0.0, 1.0]	0 [0.0, 1.0]	0	13.33 ms
DP (chunk=16)	83 [79.0, 86.4]	5 [3.3, 7.6]	78	660.48 ms
DP (chunk=8)	81 [76.9, 84.5]	6 [4.1, 8.8]	75	619.36 ms
DP (chunk=4)	82 [77.9, 85.5]	2 [1.0, 3.9]	80	630.00 ms
DP (chunk=2)	82 [77.9, 85.5]	0 [0.0, 1.0]	82	636.46 ms
DP (chunk=1)	76 [71.6, 79.9]	0 [0.0, 1.0]	76	643.10 ms
<b>ToolHang</b>				
BC-TF	0 [0.0, 1.0]	0 [0.0, 1.0]	0	12.97 ms
BC-TF-GMM	0 [0.0, 1.0]	0 [0.0, 1.0]	0	12.65 ms
BC-Gaussian	0 [0.0, 1.0]	0 [0.0, 1.0]	0	8.34 ms
BC	0 [0.0, 1.0]	0 [0.0, 1.0]	0	8.13 ms
BC-GMM	0 [0.0, 1.0]	0 [0.0, 1.0]	0	9.29 ms
BC-VAE	0 [0.0, 1.0]	0 [0.0, 1.0]	0	8.33 ms
DP (chunk=16)	46 [41.2, 50.9]	32 [27.6, 36.7]	14	572.72 ms
DP (chunk=8)	30 [25.7, 34.7]	29 [24.8, 33.6]	1	588.40 ms
DP (chunk=4)	35 [30.5, 39.8]	26 [21.9, 30.5]	9	572.12 ms
DP (chunk=2)	33 [28.6, 37.8]	2 [1.0, 3.9]	31	573.30 ms
DP (chunk=1)	31 [26.7, 35.7]	0 [0.0, 1.0]	31	566.81 ms

Table 3: Detailed performance on Robomimic tasks under synchronous and asynchronous simulation, evaluated over 400 episodes. Success rates (%) are shown with 95% Wilson score confidence intervals in brackets.

402 **B LIBERO full results**

403 We provide detailed per-policy results on the LIBERO benchmark [4] across four tasks: **Spatial**,  
 404 **Object**, **Goal**, and **Long**, as well as their average.

		Task Success Rate (%)					Performance drop (sync → async)	Rel. drop (%)	IAR ↑	Latency ↓
		Sync / Async								
		Spatial	Object	Goal	Long	Avg.				
SmolVLA	450M	90 / 56	96 / 52	92 / 67	71 / 50	87.2 / 56.2	-31.0	<b>35.6</b>	64.5	9.31 ms
X-VLA	0.9B	93 / 83	98 / 95	99 / 94	98 / 92	97.0 / 91.0	-6.0	6.2	93.8	14.32 ms
MolmoAct2	5B	98 / 96	100 / 98	98 / 96	97 / 93	98.2 / 95.8	-2.5	<b>2.4</b>	97.6	16.14 ms
$\pi_0$	4B	90 / 52	86 / 68	95 / 72	73 / 44	86.0 / 59.0	-27.0	31.4	68.6	44.46 ms
$\pi_{0.5}$	4B	97 / 68	99 / 65	98 / 74	96 / 77	97.5 / 71.0	-26.5	27.2	72.8	66.73 ms

Table 4: Synchronous and asynchronous simulation performance on the LIBERO benchmark [4].

405 **C Real-World Evaluation Details**

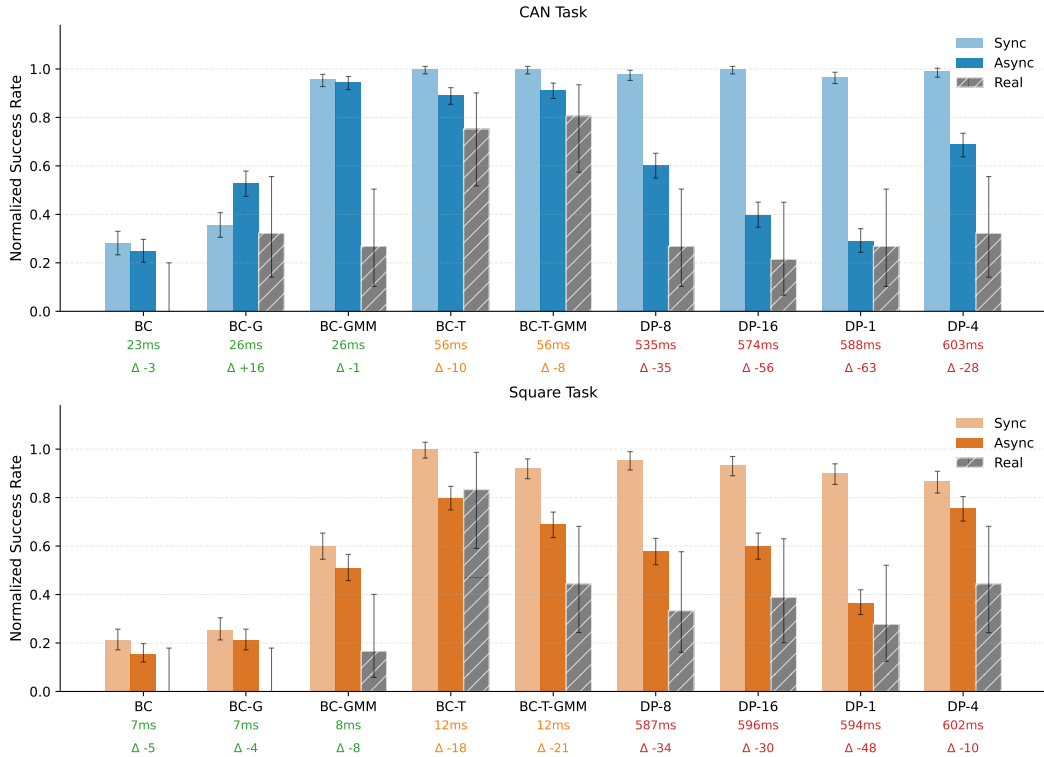


Figure 6: Per-policy success rates under synchronous, asynchronous, and real-world evaluation on the **Can** and **Square** tasks. Values are min-max normalized per task. Error bars indicate 95% Wilson score confidence intervals (400 trials for simulation, 20 for real world).

Policy	Sync	Async	$\Delta$	Real	Inference Latency
BC-TF	98 [96.1, 99.0]	88 [84.4, 90.8]	10	75 [53.1, 88.8]	55.76 ms
BC-TF-GMM	98 [96.1, 99.0]	90 [86.7, 92.6]	8	80 [58.4, 91.9]	56.01 ms
BC-Gaussian	38 [33.4, 42.8]	54 [49.1, 58.8]	-16	35 [18.1, 56.7]	26.07 ms
BC	31 [26.7, 35.7]	28 [23.8, 32.6]	3	5 [0.9, 23.6]	22.76 ms
BC-GMM	94 [91.2, 95.9]	93 [90.1, 95.1]	1	30 [14.5, 51.9]	25.94 ms
BC-VAE	87 [83.3, 89.9]	72 [67.4, 76.2]	15	-	23.27 ms
DP (chunk=16)	98 [96.1, 99.0]	42 [37.3, 46.9]	56	25 [11.2, 46.9]	574.39 ms
DP (chunk=8)	96 [93.6, 97.5]	61 [56.1, 65.7]	35	30 [14.5, 51.9]	534.97 ms
DP (chunk=4)	97 [94.8, 98.3]	69 [64.3, 73.3]	28	35 [18.1, 56.7]	602.67 ms
DP (chunk=2)	97 [94.8, 98.3]	10 [7.4, 13.3]	87	-	623.06 ms
DP (chunk=1)	95 [92.4, 96.7]	32 [27.6, 36.7]	63	30 [14.5, 51.9]	587.53 ms

Table 5: Performance on the Robomimic **Can** task under synchronous simulation, asynchronous simulation, and real-world evaluation. Success rates (%) are shown with 95% Wilson score confidence intervals in brackets.

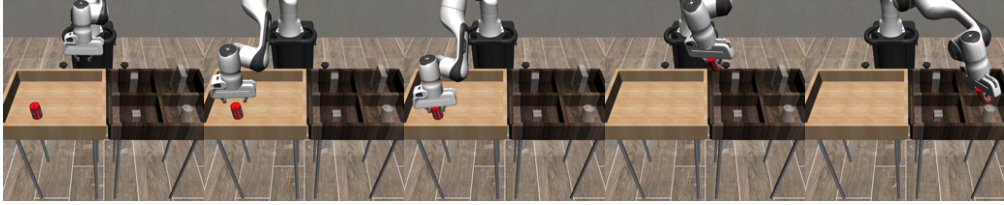
Policy	Sync	Async	$\Delta$	Real	Inference Latency
BC-TF	90 [86.7, 92.6]	72 [67.4, 76.2]	18	75 [53.1, 88.8]	12.34 ms
BC-TF-GMM	83 [79.0, 86.4]	62 [57.2, 66.6]	21	40 [21.9, 61.3]	12.38 ms
BC-Gaussian	23 [19.1, 27.4]	19 [15.5, 23.1]	4	0 [0.0, 16.1]	7.13 ms
BC	19 [15.5, 23.1]	14 [10.9, 17.7]	5	0 [0.0, 16.1]	6.72 ms
BC-GMM	54 [49.1, 58.8]	46 [41.2, 50.9]	8	15 [5.2, 36.0]	7.75 ms
BC-VAE	28 [23.8, 32.6]	24 [20.1, 28.4]	4	-	6.79 ms
DP (chunk=16)	84 [80.1, 87.3]	54 [49.1, 58.8]	30	35 [18.1, 56.7]	595.73 ms
DP (chunk=8)	86 [82.3, 89.1]	52 [47.1, 56.9]	34	30 [14.5, 51.9]	587.12 ms
DP (chunk=4)	78 [73.7, 81.8]	68 [63.3, 72.4]	10	40 [21.9, 61.3]	602.23 ms
DP (chunk=2)	83 [79.0, 86.4]	30 [25.7, 34.7]	53	-	589.04 ms
DP (chunk=1)	81 [76.9, 84.5]	33 [28.6, 37.8]	48	25 [11.2, 46.9]	594.50 ms

Table 6: Performance on the Robomimic **Square** task under synchronous simulation, asynchronous simulation, and real-world evaluation. Success rates (%) are shown with 95% Wilson score confidence intervals in brackets.

## 406 D Simulation Demos

### 407 D.1 RoboMimic Simulation Demos

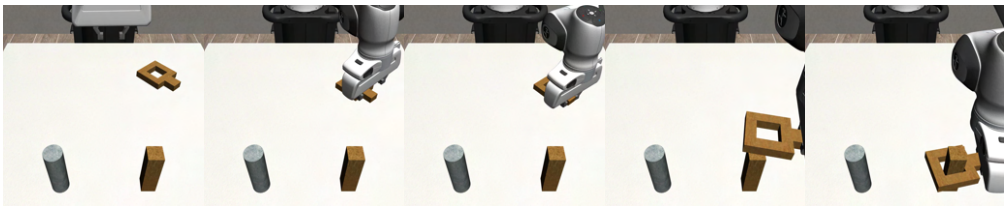
408 **Tasks.** We provide RoboMimic simulation demos on five representative manipulation tasks: **Can**,  
409 **Square**, **Lift**, **Tool Hang**, and **Transport**.



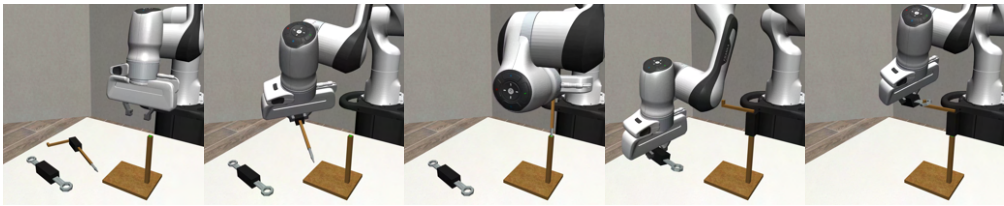
(a) **Can**: pick up a soda can from one bin and place it into the target bin.



(b) **Lift**: grasp a small cube from the table and lift it above a threshold height.



(c) **Square**: pick up a square nut and place it onto a vertical peg, requiring precise alignment.



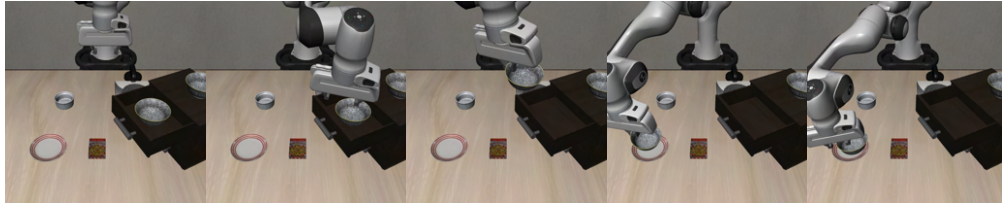
(d) **Tool Hang**: assemble a frame by inserting a hook into the base, then hang a wrench on the hook: a long-horizon, high-precision task.



(e) **Transport**: a bimanual task where two arms coordinate to transfer a hammer from a closed container to a target bin.

Figure 7: Rollout visualizations on the five RoboMimic tasks.

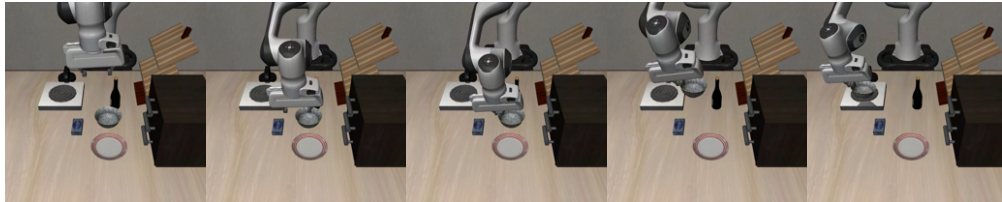
410 **D.2 LIBERO Simulation Demos**



(a) **LIBERO-Spatial**: pick-and-place tasks that require grounding spatial relations between objects in a tabletop scene.



(b) **LIBERO-Object**: pick-and-place tasks that require identifying and manipulating the correct household object in a kitchen scene.



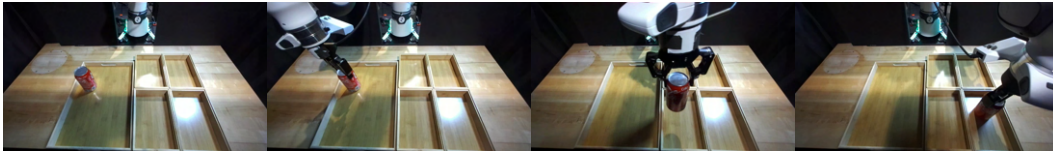
(c) **LIBERO-Goal**: short-horizon manipulation tasks where the policy must execute the goal specified by the language instruction.



(d) **LIBERO-Long**: long-horizon kitchen manipulation tasks that require chaining multiple sub-skills to completion.

Figure 8: Rollout visualizations on the four LIBERO task suites.

411 **E Real-world Demos**



(a) **Can**: pick up a soda can from one bin and place it into the target bin. (front camera view)



(b) **Square**: pick up a square nut and place it onto a vertical peg, requiring precise alignment. (external camera view; not used for training)

Figure 9: Rollout visualizations on the two real-world Robomimic [3] **Can** and **Square** tasks.