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

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

Python Song[1] **Hyeonho Oh**[2] **Ishika Singh**[2]
Jesse Thomason[2] **Ajay Mandlekar**[3] **Siddharth Ancha**[4]
[1]Columbia University [2]University of Southern California [3]NVIDIA [4]UC Berkeley

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

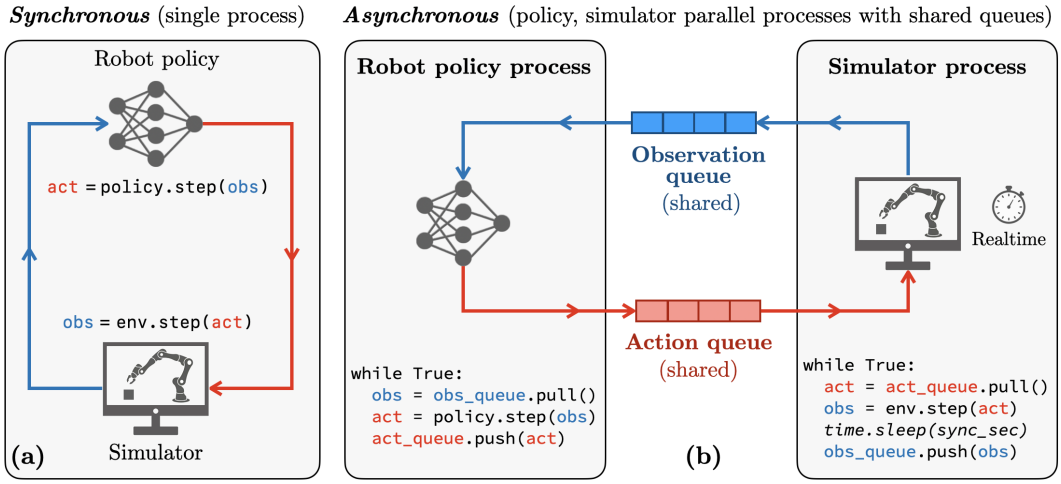


Figure 1: **(a)** Conventional *synchronous* simulation alternates between a policy inference step and a simulation step in a single process where the simulator is frozen during policy inference. **(b)** *Asynchronous* simulation runs the policy and simulator in separate parallel processes that communicate 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.

1 Introduction and related work

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

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

What about real-world benchmarks? In contrast to simulation, real-world benchmarks like RoboArena [21], AutoEval [22] and ManipulationNet [23] directly evaluate robot policies on real robots. While they are inherently asynchronous and do not suffer from the "sync-to-async" gap, real-world benchmarking is cumbersome as robots can be expensive to maintain. Furthermore, real-world experiments are challenging to scale and hard to reproduce. On the flip side, simulation benchmarks can be scaled to a large diversity of scenes and objects, are reproducible, and can be run from the comfort of one's personal computer without needing access to physical robots. Indeed, simulation and benchmarking have enabled tremendous progress in computer vision [24] and reinforcement learning [17, 25–28]. Prior works on simulation benchmarks focus on reducing well-known "sim-to-real gap" in photorealism of image rendering [20, 29, 30], sensor simulation and physics accuracy [9]. This progress is orthogonal to our work. We posit that simulation benchmarks need not suffer from yet another, artificially introduced "sync-to-async" gap: the simulator and policy can and should be run asynchronously. *Asynchronous* simulation benchmarking provides an important middle ground: it allows us to evaluate the speed-accuracy tradeoff of robot policies that is critical to real-world deployment, while being accessible, scalable and reproducible. In Sec. 3, we show that the performance in asynchronous simulation is significantly more correlated with real-world performance of the same task (see Fig. 2), compared to the performance predicted by conventional synchronous simulation.

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

(a) `Can`-sim (b) `Can`-real (c) `Square`-sim (d) `Square`-real

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.

simulation is that it naturally reflects the policy's latency in downstream performance, thereby unifying the two in a single metric. Asynchronous simulation is particularly well-suited to evaluate methods that are designed to reduce latency and improve smoothness of policies, such as Real-Time Chunking [31, 32], VLASH [33] and streaming policies [34]. In Sec. 3, we find that the higher the latency of a policy, the worse is the performance estimate of conventional synchronous simulation.

Useful for debugging: We found that asynchronous simulation is an extremely useful debugging tool that clearly visualizes the latency of robot policies. For example, it is common for diffusion policies [35] to "jitter" at the boundaries of action chunks where the diffusion inference takes place. This jittery motion profile was clearly visible in `AsyncRoboSim` matching real-world behavior. This allows users to debug and fix latency-related issues in their robot policies even before deploying them in the real world. In contrast, the motion in synchronous simulation appears deceptively smooth.

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

2 Method

Parallel architecture and instantiation in *robosuite*: As shown in Fig. 1, the `AsyncRoboSim` architecture is organized into two parallel processes: one that runs the policy and the other that runs the simulator. We implement our approach in `robosuite` [2]: a MuJoCo-based [7] simulation framework that many [3–6] robotic benchmarks build upon. We implement our approach using Python's `multiprocessing` [36] library, and interface with the MuJoCo physics engine [7] via the `mujoco` Python library. The policy and simulator processes communicate via shared observation and action queues that are thread-safe and multiprocessing-safe. This architecture decouples the speed of the policy from the speed of the simulator. Since neither process waits on each other, the simulator is not throttled by slow policies. Each process runs in a loop until a shutdown signal is received.

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

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

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

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

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

3 Experiments

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

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

[1] https://anonymous.4open.science/r/realtime-robosuite/robosuite/environments/async_env.py#L343
[2] https://anonymous.4open.science/r/realtime-robosuite/robosuite/environments/async_env.py#L315-L323

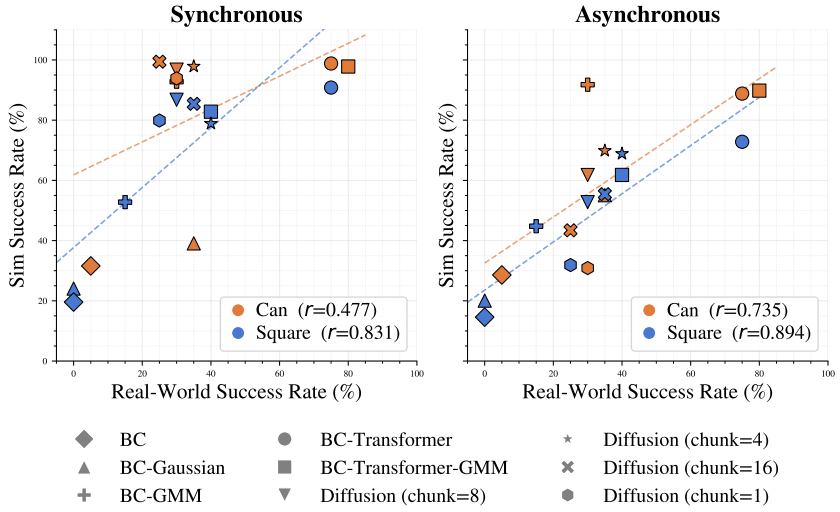


Figure 3: Linear correlation between simulation and real-world success rates across all evaluated policies under synchronous and asynchronous conditions.

Task	Condition	Pearson r	Spearman ρ	Pairwise Acc.
`Can`	Synchronous sim performance	0.477	0.485	0.724
	Asynchronous sim performance	0.735	0.664	0.812
	Improvement (sync → async)	**+0.258**	**+0.179**	**+0.088**
`Square`	Synchronous sim performance	0.831	0.782	0.824
	Asynchronous sim performance	0.894	0.975	0.971
	Improvement (sync → async)	**+0.063**	**+0.193**	**+0.147**

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

intervals [37]. For LIBERO, we evaluate each model on a workstation equipped with an NVIDIA RTX 4090 GPU, using 100 trials per task suite following the standard LIBERO evaluation protocol. For real-world evaluation, we use a Franka Research 3 robot on the `Can` and `Square` tasks. Each policy uses multi-view RGB observations from a static front camera and a wrist-mounted camera, resized to $84 \times 84 \times 3$ images, and is evaluated over 20 trials. Training uses 63 demonstrations for 200 training epochs on `Can` and 120 demonstrations for 400 training epochs on `Square`. Our policy suite spans three policy families benchmarked in `robosuite` [2]: (1) MLP-based BC variants, (2) Transformer-based BC variants, and (3) Diffusion Policy. The BC variants cover deterministic, Gaussian, GMM, VAE, and transformer/transformer-GMM architectures. For diffusion policy, we train the model to predict 16 action steps and evaluate execution chunk sizes $T_a \in \{1, 2, 4, 8, 16\}$ by using only the first T_a predicted actions under both synchronous and asynchronous settings. The spread of action chunk size provides a wide variation over policy performance and latencies.

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

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

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

1. **Pearson linear correlation (r):** Given paired simulated and real-world success rates $\{(x_i, y_i)\}_{i=1}^n$ 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
$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2}\sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$
This metric measures *linear* correlation, and is sensitive to the magnitude of differences.
2. **Spearman rank-correlation (ρ):** Let k_i^{real} be rank of the i-th policy according to real-world success, and k_i^{sim} be its rank according to simulated success. The Spearman rank correlation is simply the linear correlation between ordinal ranks:
$$\rho = 1 - \frac{6\sum_{i=1}^n (k_i^{\text{real}} - k_i^{\text{sim}})^2}{n\,(n^2 - 1)}$$
This metric measures rank correlation. It is insensitive to the magnitude of performance difference. Instead, it focuses on the relative ordering of policies.
3. **Pairwise ranking accuracy (PRA):** across each possible pairs of robot policies, we check whether the binary ranking according to real-world success rate matches that of the simulated success rate. PRA is the fraction of pairs where the binary ranking matches. Given paired simulated and real-world success rates $\{(x_i, y_i)\}_{i=1}^n$ across policies, PRA is given by
$$\text{PRA} = \binom{n}{2}^{-1} \sum_{i<j} \mathbb{I}[\,\text{sign}(x_i - x_j) = \text{sign}(y_i - y_j)\,]$$
where $\mathbb{I}[\cdot]$ is the indicator function. Like the Spearman correlation, this metric also measures rank correlation and is only sensitive to relative ordering of policies.

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

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

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

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

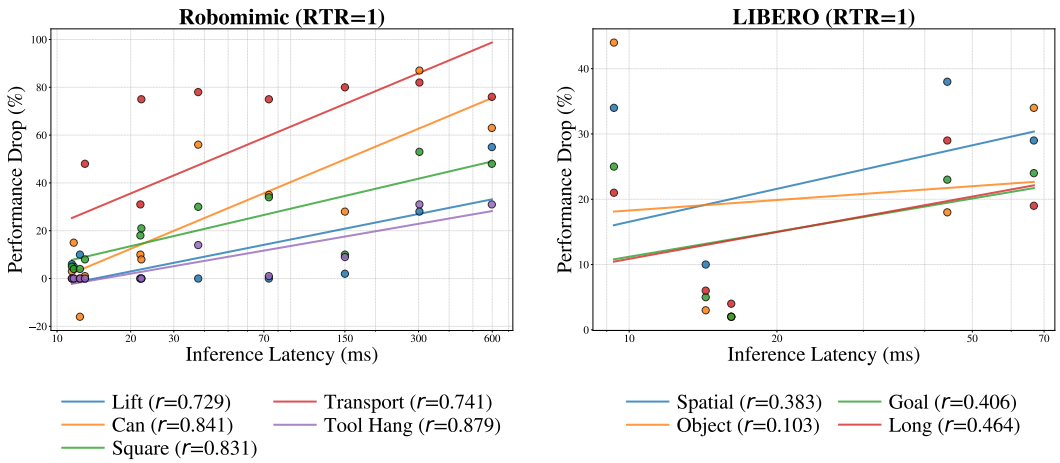


Figure 4: Performance drop (sync → async, %) vs. inference latency across all evaluated policies on Robomimic (left) and LIBERO (right). Each point represents a single task–policy pair colored by task. Lines show per-task log-linear regression fits with slopes annotated in the legend.

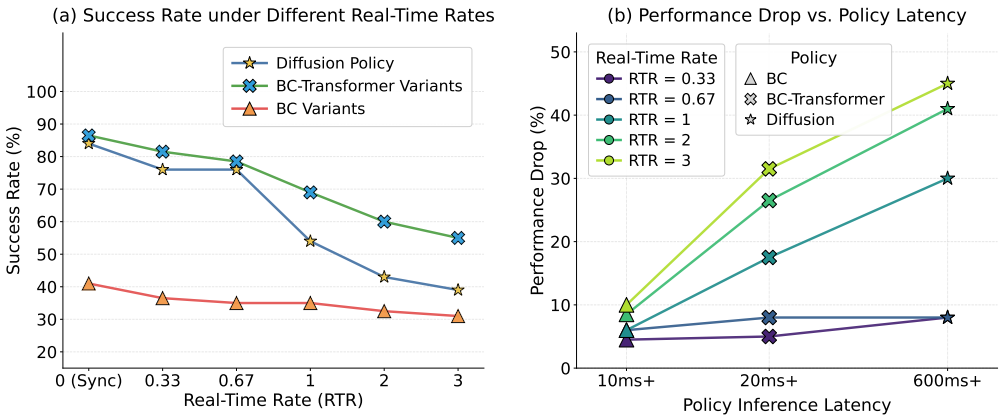


Figure 5: Analysis of success rate and "sync-to-async" gap on the Robomimic `Square` task across various robot policies. ***Left:*** Success rate reduces with increasing RTR (faster simulation), and this effect is more pronounced for higher-latency policies. ***Right:*** Performance drop from synhronous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

11 ms and 22 ms, which are comparable to the simulator control-loop timescale. As a result, their asynchronous performance largely preserves the trends observed under synchronous evaluation. For example, low-latency policies achieve similar performance on the `Can` task under synchronous and asynchronous evaluation. In contrast, diffusion policy [35] variants exhibit a substantially larger gap between synchronous and asynchronous evaluation. Although these policies achieve strong synchronous success rates, their iterative denoising process incurs much higher inference latency. Under synchronous evaluation, this computation time is effectively hidden, as the simulator waits for policy inference to complete. Under asynchronous evaluation, the simulator instead continues to advance in real time executing the most recent available action, reducing closed-loop reactivity. On the `Can` task, diffusion policy with chunk size 16 drops from 98% success under synchronous evaluation to 42% under asynchronous evaluation.

LIBERO latency results (RQ2): We further study how synchronous simulation affects vision-language-action (VLA) models, which are heavier robot policies with vision-language backbones. We select a diverse set of VLA policies that span the spectrum of capability vs. latency. SmolVLA [38] is a compact 450M-parameter model designed for efficient deployment, while X-VLA [39] and MolmoAct2 [40] represent recent high-performing open VLA models with stronger action-reasoning capabilities. We further include π_0 [41] and $\pi_{0.5}$ [42] as widely adopted generalist robot policies.

In Fig. 4 (right), we find that the correlation between inference latency and performance drop for VLAs on LIBERO is positive, but is weaker than the robot control policies in Robomimic. Asynchronous robustness may also depend on policy-level properties, such as temporal stability, action correction, and the ability to recover from stale commands. We find that X-VLA retains 93.8% of its performance and MolmoAct2 retains 97.6% of its performance going from synchronous to asynchronous simulation. In contrast, π_0 retains only 68.6%. We observe self-correction behavior from MolmoAct2 in our asynchronous simulator. *Importantly, this behavior is not visible under synchronous simulation*, and is only visible when accounting for the asynchronous interaction between the policy and the environment. Full results on the LIBERO benchmark are provided in App. B.

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

4 Limitations

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

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

5 Conclusion

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

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[2] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, Y. Zhu, and K. Lin. `robosuite`: a modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[3] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. `robomimic`: what matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

[4] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

[5] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu. RoboCasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

[6] S. Nasiriany, S. Nasiriany, A. Maddukuri, and Y. Zhu. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. *arXiv preprint arXiv:2603.04356*, 2026.

[7] 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.

[8] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

[9] R. Tedrake et al. Drake: Model-based design and verification for robotics, 2019.

[10] G. Authors. Genesis: A Generative and Universal Physics Engine for Robotics and Beyond, December 2024. URL `https://github.com/Genesis-Embodied-AI/Genesis`.

[11] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[12] S. Tao, F. Xiang, A. Shukla, Y. Qin, X. Hinrichsen, X. Yuan, C. Bao, X. Lin, Y. Liu, T.-k. Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

[13] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, et al. ORBIT: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023.

[14] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[15] Y. Kim, W. Pumacay, O. Rayyan, M. Argus, W. Han, E. VanderBilt, J. Salvador, A. Deshpande, R. Hendrix, S. Jauhri, et al. MolmoSpaces: A large-scale open ecosystem for robot navigation and manipulation. *arXiv preprint arXiv:2602.11337*, 2026.

[16] TRI LBM Team, J. Barreiros, A. Beaulieu, A. Bhat, R. Cory, E. Cousineau, H. Dai, C.-H. Fang, K. Hashimoto, M. Z. Irshad, et al. LBM Eval: A careful examination of large behavior models for multitask dexterous manipulation. *arXiv preprint arXiv:2507.05331*, 2025. URL `https://github.com/ToyotaResearchInstitute/lbm_eval`.

[17] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

[18] C. Li, R. Zhang, J. Wong, C. Gokmen, S. Srivastava, R. Martín-Martín, C. Wang, G. Levine, M. Lingelbach, J. Sun, et al. BEHAVIOR-1K: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023.

[19] C. Sferrazza, D.-M. Huang, X. Lin, Y. Lee, and P. Abbeel. Humanoidbench: Simulated humanoid benchmark for whole-body locomotion and manipulation. *arXiv preprint arXiv:2403.10506*, 2024.

[20] Y. Jangir, Y. Zhang, P.-C. Lo, K. Yamazaki, C. Zhang, K.-H. Tu, T.-W. Ke, L. Ke, Y. Bisk, and K. Fragkiadaki. RobotArena ∞: Scalable Robot Benchmarking via Real-to-Sim Translation. *arXiv preprint arXiv:2510.23571*, 2025.

[21] P. Atreya, K. Pertsch, T. Lee, M. J. Kim, A. Jain, A. Kuramshin, C. Eppner, C. Neary, E. Hu, F. Ramos, et al. RoboArena: Distributed real-world evaluation of generalist robot policies. *arXiv preprint arXiv:2506.18123*, 2025.

[22] Z. Zhou, P. Atreya, Y. L. Tan, K. Pertsch, and S. Levine. AutoEval: Autonomous evaluation of generalist robot manipulation policies in the real world. *arXiv preprint arXiv:2503.24278*, 2025.

[23] Y. Chen, K. Kimble, E. H. Adelson, T. Asfour, P. Chanrungmaneekul, S. Chitta, Y. Chitambar, Z. Chen, K. Goldberg, D. Kragic, et al. ManipulationNet: An infrastructure for benchmarking real-world robot manipulation with physical skill challenges and embodied multimodal reasoning. *arXiv preprint arXiv:2603.04363*, 2026.

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[25] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of artificial intelligence research*, 47:253–279, 2013.

[26] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

[27] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[28] W. Zhao, J. P. Queralta, and T. Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE, 2020.

[29] X. Li, K. Hsu, J. Gu, K. Pertsch, O. Mees, H. R. Walke, C. Fu, I. Lunawat, I. Sieh, S. Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

[30] A. Jain, M. Zhang, K. Arora, W. Chen, M. Torne, M. Z. Irshad, S. Zakharov, Y. Wang, S. Levine, C. Finn, et al. Polaris: Scalable real-to-sim evaluations for generalist robot policies. *arXiv preprint arXiv:2512.16881*, 2025.

[31] K. Black, M. Galliker, and S. Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38:33383–33407, 2026.

[32] K. Black, A. Z. Ren, M. Equi, and S. Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025.

[33] J. Tang, Y. Sun, Y. Zhao, S. Yang, Y. Lin, Z. Zhang, J. Hou, Y. Lu, Z. Liu, and S. Han. VLASH: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.

[34] S. Jiang, X. Fang, N. Roy, T. Lozano-Pérez, L. P. Kaelbling, and S. Ancha. Streaming flow policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories. *arXiv preprint arXiv:2505.21851*, 2025.

[35] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

[36] Python Software Foundation. `multiprocessing` — process-based parallelism. Python Standard Library documentation, 2024. URL `https://docs.python.org/3/library/multiprocessing.html`.

[37] E. B. Wilson. Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, 22(158):209–212, 1927.

[38] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[39] J. Zheng, J. Li, Z. Wang, D. Liu, X. Kang, Y. Feng, Y. Zheng, J. Zou, Y. Chen, J. Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

[40] H. Fang, J. Duan, D. Clay, S. Wang, S. Liu, W. Huang, X. Fan, W.-C. Tsai, S. Chen, Y. R. Wang, et al. MolmoAct2: Action Reasoning Models for Real-world Deployment. *arXiv preprint arXiv:2605.02881*, 2026.

[41] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A Vision-Language-Action Flow Model for General Robot Control. *arXiv preprint arXiv:2410.24164*, 2024.

[42] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, et al. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. *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

A Robomimic full results

We provide detailed per-policy results on the Robomimic benchmark [3] across all five tasks: `Lift`, `Can`, `Square`, `Transport`, and `ToolHang` in Table 3. All values are success rates (%) evaluated over 400 episodes under synchronous and asynchronous settings. 95% Wilson score confidence intervals [37] are reported in brackets.

		Task Success Rate (%) Sync / Async						Perf. drop	Rel. drop (%)	**Latency** ↓
		`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	**24.2** / 24.6	+0.4	**−1.65**	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 / **22.0**	−54.6	**71.28**	597.80 ms
9	DP (chunk=4)	100 / 98	97 / 69	78 / 68	82 / 2	35 / 26	78.4 / **52.6**	−25.8	32.91	150.05 ms
10	DP (chunk=16)	100 / 100	98 / 42	84 / 54	83 / 5	46 / 32	**82.2** / 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	**−56.2**	71.14	302.40 ms

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

Policy	**Sync**	**Async**	**Δ**	**Inference Latency**
`Can`				
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
`Square`				
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	Δ	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
`Lift`				
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
`Transport`				
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
`ToolHang`				
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.

B LIBERO full results

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

		Task Success Rate (%) Sync / Async					Performance drop (sync → async)	Rel. drop (%)	**IAR** ↑	**Latency** ↓
		`Spatial`	`Object`	`Goal`	`Long`	Avg.				
SmolVLA	450M	90 / 56	96 / 52	92 / 67	71 / 50	87.2 / 56.2	−31.0	**35.6**	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	**2.4**	97.6	16.14 ms
π_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].

C Real-World Evaluation Details

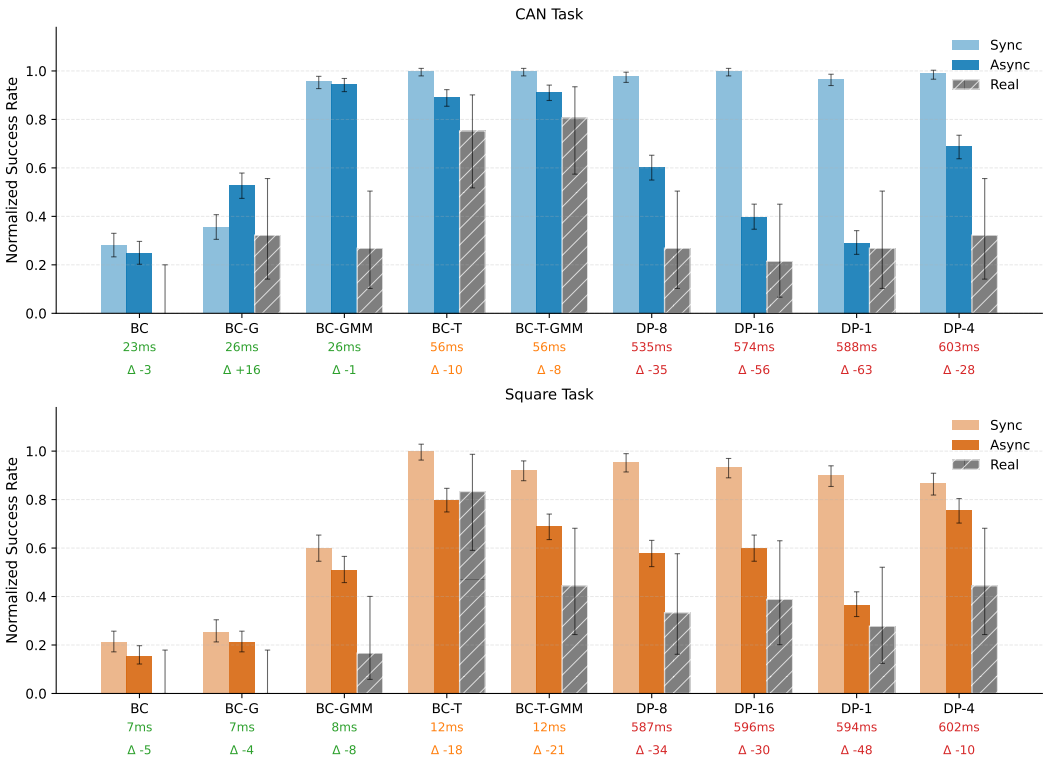


Figure 6: Per-policy success rates under synchronous, asynchronous, and real-world evaluation on the `Can` and `Square` tasks. Values are min-max normalized per task. Error bars indicate 95% Wilson score confidence intervals (400 trials for simulation, 20 for real world).

Policy	Sync	Async	Δ	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.

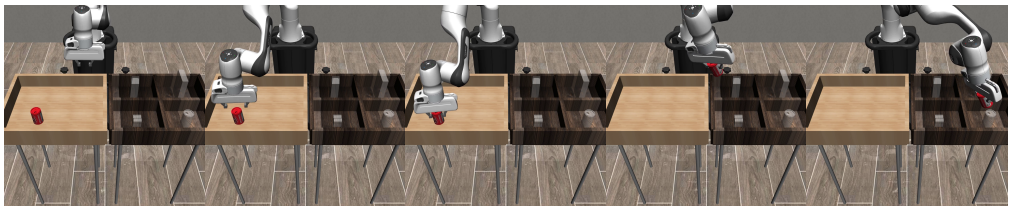
Policy	Sync	Async	Δ	Real	Inference Latency
BC-TF	90 [86.7, 92.6]	72 [67.4, 76.2]	18	75 [53.1, 88.8]	12.34 ms
BC-TF-GMM	83 [79.0, 86.4]	62 [57.2, 66.6]	21	40 [21.9, 61.3]	12.38 ms
BC-Gaussian	23 [19.1, 27.4]	19 [15.5, 23.1]	4	0 [0.0, 16.1]	7.13 ms
BC	19 [15.5, 23.1]	14 [10.9, 17.7]	5	0 [0.0, 16.1]	6.72 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.

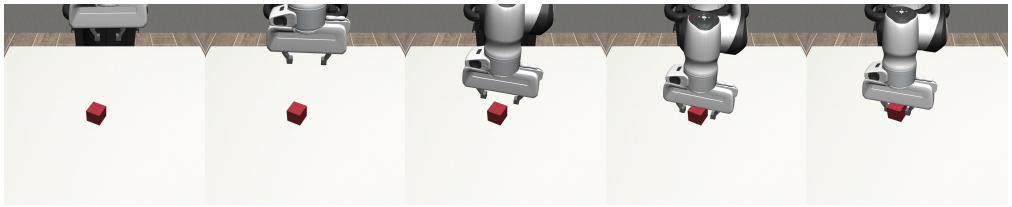
D Simulation Demos

D.1 RoboMimic Simulation Demos

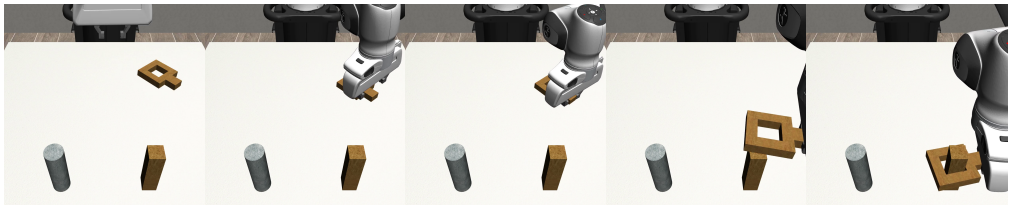
Tasks. We provide RoboMimic simulation demos on five representative manipulation tasks: `Can`, `Square`, `Lift`, `Tool Hang`, and `Transport`.



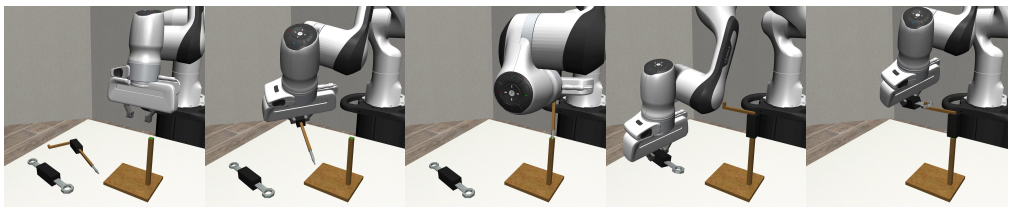
(a) **Can**: pick up a soda can from one bin and place it into the target bin.



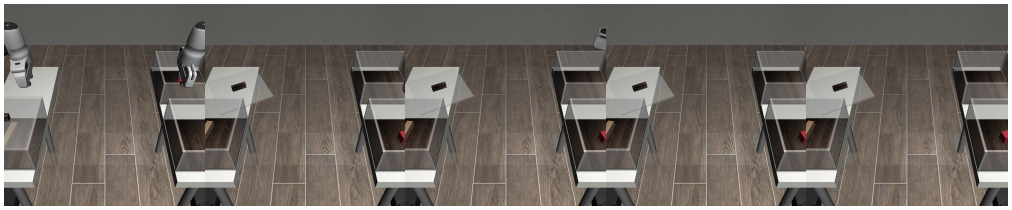
(b) **Lift**: grasp a small cube from the table and lift it above a threshold height.



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



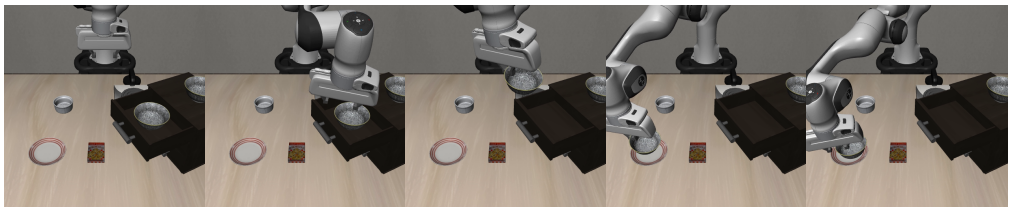
(d) **Tool Hang**: assemble a frame by inserting a hook into the base, then hang a wrench on the hook: a long-horizon, high-precision task.



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

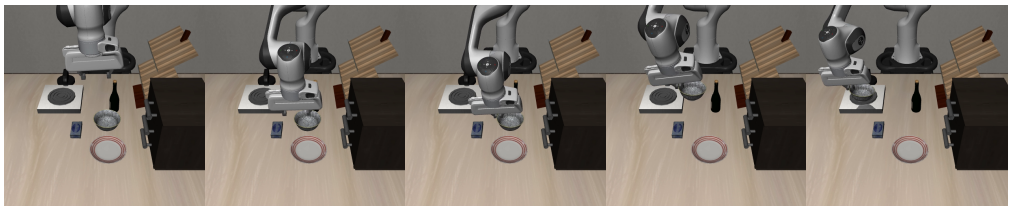
Figure 7: Rollout visualizations on the five RoboMimic tasks.

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.



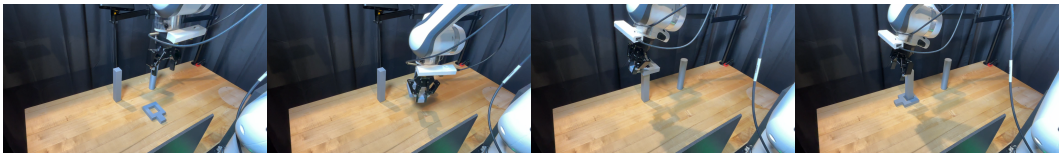
(c) `LIBERO-Goal`: short-horizon manipulation tasks where the policy must execute the goal specified by the language instruction.

(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.

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.