RETIMING ROBOT DEMONSTRATIONS WITH A LEARNED SPEED PRIOR

Anonymous authors
Paper under double-blind review

ABSTRACT

The pace of a robot demonstration is often determined by the teleoperator rather than by the task itself. As a result, the same motion can appear at widely different speeds across demonstrations, and policies trained on such data may inherit operator-specific timing along with task behavior. We propose to learn a speed prior directly from large-scale demonstration data. Given a task instruction and a motion path, it predicts a distribution over execution speed along the path, from which a chosen quantile defines a consistent clock for retiming each demonstration. Existing methods derive the acceleration signal from the very data they accelerate, so each new task costs another fit, annotation, or pair of predefined rates. A single trained speed prior instead covers every task in a corpus. We fit a speed prior on both robot teleoperation data and handheld demonstrations in which a human arm, rather than a robot arm, moves the gripper, and use it to retime robot demonstrations to the handheld pace before fine-tuning a vision-language-action policy. For execution, we pair the policy with a real-time optimal-control layer that tracks predicted action chunks under robot constraints using the predicted pace. On 120 RoboLab tasks, policies fine-tuned on our learned clocks succeed more often and move with up to half the jerk compared with policies fine-tuned on the original demonstration timing and uniformly accelerated to the same speed. On a real robot, our policy stacks cups in $3.43\,\mathrm{s}$, $1.65\times$ faster than baseline's $5.67\,\mathrm{s}$ while succeeding in all 30 trials. Videos and further material are available at `https://anonymous.4open.science/w/anony_repo-882B/`.

1 INTRODUCTION

The pace of robot demonstrations depends on both the collection setup and the operator. Teleoperating a robot through an interface is slower than directly moving a handheld gripper (Chi et al., 2024), and novice operators often hesitate or pause where experienced operators do not. Policies trained to imitate these demonstrations inherit the operators' pace along with the task behavior, often executing more slowly than necessary (Guo et al., 2025; Arachchige et al., 2025).

Prior work speeds up such policies by retiming their demonstrations before or during training, or by accelerating their execution (Guo et al., 2025; Kim et al., 2026a; Sanchez et al., 2026; Nam et al., 2025; Jing et al., 2026; Hu et al., 2026; Arachchige et al., 2025; Kim et al., 2026c). These methods shorten motion segments that require less precision, train the policy at a range of speeds, or execute the motion as fast as the robot's limits allow. They typically adjust the original demonstration timing based on task requirements, external speed commands, or robot limits. Few instead learn the pace from demonstrators who perform similar motions faster.

We instead draw the pace from such demonstrators, for example those using a handheld gripper or experienced with the task, and give it to the demonstrations that the policy learns from. Their demonstrations need not follow the same paths, so we do not copy their timing but learn a speed prior, a model of how fast they move along a given path. Given the local shape of an end-effector path and the task instruction, the prior predicts a distribution over the speed at which demonstrators traverse each part of the path. It is trained once on both kinds of demonstrations across all their tasks, with a token that marks the kind of each demonstration, so reading a path with the token of the faster demonstrators predicts how fast they would traverse it. A quantile of this distribution defines a clock that keeps the path of a demonstration but replaces its timing. We then fine-tune a

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

policy on demonstrations retimed with the median clock and let it also predict tempo channels, the pace of higher quantiles, so that it can be sped up at deployment.

The predicted pace reflects human motion and may therefore exceed the robot's velocity, acceleration, or torque limits. Joint PD control, commonly used to execute such policies, does not account for these limits in advance, so fast targets can cause tracking lag or trigger safety reflexes (Khazatsky et al., 2024; Physical Intelligence, 2025). Consecutive action chunks may also disagree at their boundaries, and each new chunk must accommodate the pace of the ongoing motion. We therefore introduce a controller that treats the predicted pace as an upper bound and executes each chunk as fast as the robot's limits permit, slowing down only where a limit becomes active. When a new chunk arrives, the controller smoothly connects it to the ongoing motion, preserving continuity in position, velocity, and acceleration while carrying the execution clock forward.

We train the speed prior on teleoperated demonstrations from DROID and handheld demonstrations from FastUMI-100K, and use it to retime DROID to the handheld pace before fine-tuning the released $\pi_{0.5}$ DROID policy (Khazatsky et al., 2024; Liu et al., 2025; Intelligence et al., 2025; Physical Intelligence, 2025). On 120 tasks of the RoboLab simulation benchmark, this policy succeeds more often than a uniformly accelerated policy at the same speed-up, and at $2.8\times$ it does so with half the jerk (Yang et al., 2026c). Training only on fast demonstrations does not achieve this. A policy trained on the fastest DROID episodes alone, although their pace is $1.60\times$ that of all episodes, runs just $1.04\times$ faster than the policy trained on all of them. We observe the same on a real robot, where we adapt the prior to handheld demonstrations collected with a YUBI gripper and retime them to the pace of the shortest quarter of the episodes (Ohkawa et al., 2026). Our policy stacks cups in $3.43\,\mathrm{s}$ instead of $5.67\,\mathrm{s}$ and succeeds in all 30 trials, whereas a policy trained only on this quarter succeeds in 4 of 30 trials. On the same three-cup pyramid, SAIL reports $14.71\,\mathrm{s}$ at a success rate of 0.40 for a diffusion policy trained on teleoperated demonstrations and accelerated by $1.76\times$ (Arachchige et al., 2025), so our policy stacks them $4.3\times$ faster and succeeds in every trial. Finally, executing a human pace requires a controller that respects the robot's limits. With our controller in place of joint PD control, even the policy trained on the recorded timing folds the towel in 87% of trials instead of 3%.

2 Related work

Retiming demonstrations. A line of work changes the timing of demonstrations before policy training. Arc-length warping reparameterizes each trajectory by path length and leaves the execution speed to be chosen afterward (Braglia et al., 2025). ISR keeps a subset of frames spaced at roughly equal information, measured by velocity and acceleration (Yang et al., 2026b). Other methods downsample the segments that do not require precision. DemoSpeedup finds them by the action entropy of a proxy policy trained on the task (Guo et al., 2025). ESPADA labels one seed demonstration with a vision-language model and a language model and propagates the labels to the other demonstrations of the task (Kim et al., 2026a). VOLT labels every demonstration with a vision-language model prompted by examples from other tasks (Sanchez et al., 2026). In these methods, the timing follows fixed criteria or is compressed by factors chosen by the user.

Training policies at different speeds. Other methods retime demonstrations to a range of speeds during training and differ in how they choose the executed speed. SpeedAug trains the policy on demonstrations accelerated at random rates and then fine-tunes it with reinforcement learning in the task environment (Nam et al., 2025). TempoVLA conditions a vision-language-action policy on a speed command, which a vision-language model chooses at deployment (Jing et al., 2026). AutoSpeed learns a per-timestep speed by selecting, among candidates retimed at a discrete set of ratios, the one that best trades the policy's own prediction error against its prediction horizon (Hu et al., 2026). In all of these methods, the new timing is derived from the demonstration itself, the task at hand or an external command. In contrast, we draw the pace from other demonstrators who perform similar motions faster, and learn it as a speed prior shared across tasks.

Executing policies faster. Learned policies are usually deployed by sending each predicted action as a position target to a PD controller, one action per time step of the demonstrations, which limits how much faster they can run (Khazatsky et al., 2024; Physical Intelligence, 2025). SAIL predicts whether the current motion is critical, using labels obtained by clustering waypoints of

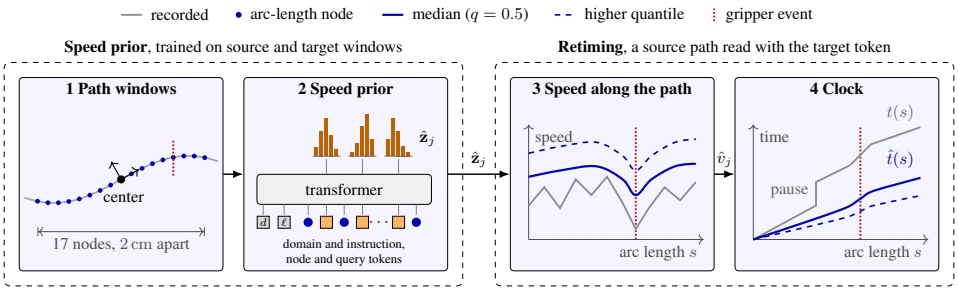

Figure 1: Learning the pace from demonstrations. (1) Paths are resampled every 2 cm of arc length and read in windows of 17 nodes. (2) The prior predicts a distribution over the log speed of every interval. (3) Reading a source path with the target token gives a quantile of the speed along the path. (4) Integrating this speed gives the clock.

the demonstrations, and scales each step duration by one of two factors (Arachchige et al., 2025). Realtime-VLA V2 retimes each chunk under velocity bounds with speeds from a model trained on the speed commands an operator gives during rollouts (Yang et al., 2026a). RACE retimes each chunk by time-optimal path parameterization, as fast as the robot's velocity and torque limits allow (Pham & Pham, 2018; Kim et al., 2026c). Consecutive chunks are reconciled inside the policy or by the executor, as in VLA-RAIL, which joins them with quintic splines but scales their timing uniformly (Black et al., 2025a;b; Zhao et al., 2025). Other policies output parameters of their controller, such as the stiffness of an impedance controller (Martín-Martín et al., 2019; Kamijo et al., 2024; Hou et al., 2025; Kim et al., 2026b). Our policy likewise outputs its pace, and our controller follows this pace within the robot's dynamic limits and joins consecutive chunks without a jump in position, velocity or acceleration.

3 Learning the Pace from Demonstrations

3.1 Paths and Clocks

We learn a speed prior that predicts how fast target demonstrators traverse each part of a path, and retime source demonstrations with a clock derived from it (Figure 1). A demonstration records end-effector positions $\mathbf{p}_k \in \mathbb{R}^3$, orientations $R_k \in SO(3)$ and gripper openings g_k at timestamps t_k, for $k = 0, \dots, K$. We measure progress along the path with an increment that combines translation and rotation,

$$\Delta s_k = \sqrt{\|\mathbf{p}_{k+1} - \mathbf{p}_k\|^2 + \rho^2 \, \theta\big(R_k^\top R_{k+1}\big)^2},$$

where $\theta(\cdot)$ is the angle of a relative rotation and ρ is a length scale that converts radians to meters (Park, 1995). We set $\rho = 0.3\,\mathrm{m/rad}$, following prior work on rigid-body trajectory segmentation (Verduyn et al., 2024). The arc length $s_k = \sum_{j<k} \Delta s_j$ grows only when the end effector translates or rotates, so it stays constant while the arm is stationary or only the gripper moves. The recorded timing is then a map $t(s)$ from arc length to time, with speed $v(s) = \mathrm{d}s/\mathrm{d}t$ along the path. Retiming replaces $t(s)$ with a clock $\hat{t}(s)$ and keeps the path.

3.2 Training Data

Source and target demonstrations. Source demonstrations are the ones we retime, and target demonstrations show the pace that they should follow. The two may come from different collection setups, such as robot teleoperation and handheld demonstrations, in which a person moves a gripper by hand (Chi et al., 2024). They may also come from the same setup when only some demonstrators move at the desired pace. Each demonstration carries a domain label that marks it as source or target. We express handheld poses at the reference point and axes of the robot data, so that arc length is computed in the same way, and remove handheld episodes whose tracking jumps (Appendix A).

Path windows. We resample each path at nodes spaced $\Delta = 2\,\mathrm{cm}$ apart in arc length, the spacing used with this metric in prior work (Verduyn et al., 2024). The prior reads windows of 17 nodes, eight on each side of a center node, spanning $32\,\mathrm{cm}$. Each node is described by its position and rotation vector in the frame of the center node and by its absolute gripper opening. The relative pose makes windows comparable across domains and setups (Chi et al., 2024).

Speed targets. The 17 nodes bound 16 intervals. Each node takes the time at which the recorded path first reaches its arc length, and the first node of an episode the time at which the arm leaves rest. The target for interval j is the log speed $y_j = \log(\Delta/\Delta t_j)$, where Δt_j is the time between its two nodes, including any pauses.

3.3 Speed prior

Architecture. The prior is a transformer of about 15M parameters trained from scratch. It reads a domain token for the domain label, an instruction token and the 17 node tokens interleaved with 16 query tokens, one for each interval. The instruction token embeds the task instruction with a frozen text encoder (Zhang et al., 2025). Each normalized node feature is low-pass filtered along the window by keeping the first nine of its 17 DCT-II coefficients, so that the prior does not read the high-frequency jitter of a particular tracking setup as a cue for pace (Appendix E). Each query token predicts a distribution over the speed of its interval (Appendix B).

Categorical objective. Different demonstrators traverse similar paths at very different speeds, so we predict a distribution over the speed of each interval rather than a single value, from which retiming can select a quantile. The prior predicts a categorical distribution $\hat{\mathbf{z}}_j$ over $B = 256$ equal bins of log speed between $1\,\mathrm{mm/s}$ to $5000\,\mathrm{mm/s}$, a classification that is also more robust to noisy targets than squared-error regression (Farebrother et al., 2024). The target y_j is encoded as a two-hot vector $\mathbf{z}_j$ that splits unit mass between the two nearest bin centers, and targets beyond the outermost centers are clipped to them (Farebrother et al., 2024). The prior is trained with the cross-entropy loss

$$\mathcal{L} = -\frac{1}{|\mathcal{J}|} \sum_{j \in \mathcal{J}} \mathbf{z}_j^\top \log \hat{\mathbf{z}}_j,$$

where $\mathcal{J}$ holds the intervals that lie within the episode.

3.4 Retiming

To retime a source demonstration, we read its path with the target domain token, so that the clock follows the pace that the prior has learned from the target while the path remains that of the source. We decode the q-quantile of each predicted distribution from its cumulative distribution, interpolating linearly within a bin. Each interval is covered by up to 16 overlapping windows, and its speed $\hat{v}_j$ is the geometric mean of their quantile speeds. The clock assigns each recorded frame the time needed to reach its arc length at these speeds,

$$\hat{t}(s) = \int_0^s \frac{\mathrm{d}s'}{\hat{v}(s')},$$

where $\hat{v}(s)$ equals $\hat{v}_j$ on interval j. Frames at the same arc length share a timestamp, so pauses in the recording take no time on the clock.

3.5 Policy fine-tuning

Action targets. We fine-tune $\pi_{0.5}$ with joint-position action chunks on the source demonstrations retimed with the median clock ($q = 0.5$), which serves as the base clock (Intelligence et al., 2025). Observations are taken at the recorded frames, and the chunk of an observation lists the joint positions and gripper commands that the retimed demonstration reaches over the next H periods of the base clock. All policies, including the baseline, are trained on the measured joint positions of the demonstrations rather than the commanded ones, since the commands are tied to the recorded timing.

Tempo channels. To let the policy run faster at deployment, we also retime each demonstration with higher quantiles. The tempo channel of quantile q at step h is $\tau_h^{(q)} = \log\left(\delta_h/\delta_h^{(q)}\right)$, where δ_h and $\delta_h^{(q)}$ are the durations of the step on the base clock and on the clock of quantile q, and the policy predicts these channels as additional action dimensions.

4 EXECUTING THE PREDICTED PACE

Our policy predicts actions together with a pace learned from human demonstrations (Section 3.5). Executing them on a robot raises two problems. First, a human pace ignores the robot's dynamics and can exceed its velocity, acceleration or torque limits. A joint PD controller (Khazatsky et al., 2024; Physical Intelligence, 2025) builds its reference without regard to these limits and enforces them only reactively, by clamping and rate-limiting the torque or by the safety reflex, so fast targets make the robot lag or stop (Appendix F). The robot should instead follow the pace wherever its limits allow and slow down only where they bind. Second, consecutive action chunks disagree at their boundary, which real-time chunking (RTC) reduces but does not remove (Black et al., 2025a;b). Switching chunks therefore needs a transition without a jump in position, velocity or acceleration, and a timing for the new chunk. At a fixed rate, action h is due h control periods after its observation, but under the predicted pace the running motion runs ahead of or behind this schedule, so the new chunk should become due when the running motion reaches the same step.

4.1 EXECUTION CYCLE

The robot tracks a plan P, a time-parameterized reference of joint position, velocity and acceleration with gripper commands, using its own low-level controller (a 1 kHz torque controller on our Franka Research 3). Meanwhile, each cycle runs policy inference, the solver and the installation of the new plan, which take t_{vla}, t_{solve} and t_{install}. At the observation time t_o, the policy maps the camera images, the joint state $\mathbf{x}_o$, the instruction and, with RTC, the unexecuted actions of P to H joint waypoints $\mathbf{w}_h$, gripper commands c_h and tempo channels $\tau_h^{(q)}$, for the steps $h = 1, \ldots, H$ after the observation. The target pace $r_h = \exp\left(\tau_h^{(q)}\right)$, or $r_h = \alpha$ for a constant speed-up α, bounds the duration of step h from below by δ/r_h, where $\delta = 1/15\,\text{s}$ is the base clock period.

The client then reserves a switch time t_b at least $t_{\text{solve}} + t_{\text{install}}$ ahead, so the solver plans from where P will be at t_b rather than from where the robot is now, and the new plan is ready before the robot gets there. It reads two quantities off P: its state $\mathbf{s}_b = (\mathbf{x}_b, \dot{\mathbf{x}}_b, \ddot{\mathbf{x}}_b)$ at t_b, and its progress $p(t)$, the number of steps that P has advanced since t_o on its own clock, fractional between steps. By t_b, P has passed the steps up to $p(t_b)$, so the new chunk is kept from step $h_0 = \lfloor p(t_b) \rfloor + 1$ on, and $t^\star$ is the time at which P reaches that step, $p(t^\star) = h_0$ (Appendix F covers a P that ends before). The solver receives the kept waypoints $\mathbf{w}_{h_0}, \ldots, \mathbf{w}_H$, their target pace, $\mathbf{s}_b$, $t^\star$ and the robot limits, and returns a new plan P'. The robot switches to it at t_b,

$$(\mathbf{x}, \dot{\mathbf{x}}, \ddot{\mathbf{x}})(t) = \begin{cases} P(t), & t < t_b, \\ P'(t), & t \geq t_b, \end{cases} \qquad P'(t_b) = P(t_b) = \mathbf{s}_b,$$

so P' keeps the reference up to t_b and replaces the rest with a transition onto the new chunk followed by the chunk itself, without a jump in position, velocity or acceleration. The policy is queried $t_{\text{vla}} + t_{\text{solve}} + t_{\text{install}}$ before P needs a successor, so the robot never waits.

4.2 SOLVER

The solver runs three steps (Figure 2 and Algorithm 1, with details in Appendix F). DRAFT turns the new chunk into a timed path, SPLICE attaches this path to P at the switch point, and SMOOTH turns the result into one plan within all limits.

Draft. DRAFT builds a timed path $\boldsymbol{\xi}$ from the kept waypoints alone. It first fits a minimum-jerk path $\boldsymbol{\gamma}(\sigma)$ through boxes of radius η around them (Flash & Hogan, 1985). The path passes within each box rather than through each waypoint, which smooths out the jitter between predicted waypoints. It is parameterized by the base clock σ, on which step h lies at $\sigma = h\delta$, so that its shape

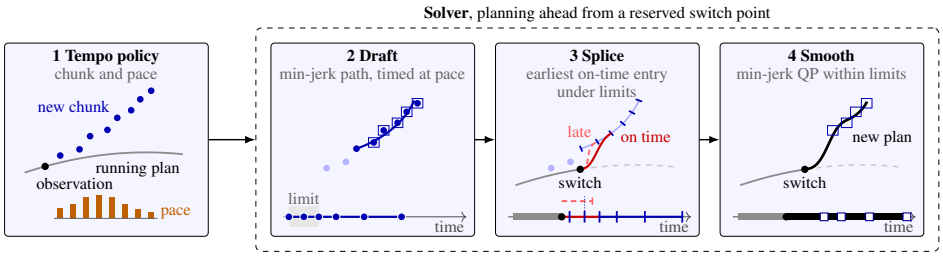

Figure 2: One execution cycle. (1) From an observation on the running plan, the policy predicts a new chunk and its pace. (2) DRAFT fits a smooth path within boxes around the kept waypoints and times it at the pace unless a limit binds. (3) SPLICE joins the running plan to this path at the earliest point it reaches on time. (4) SMOOTH solves one minimum-jerk program through the boxes at their scheduled times, within all limits, giving the new plan.

does not depend on the pace. It then plays the path back as fast as the velocity, acceleration and torque limits allow, subject to $\dot{\sigma} \leq r_h$ on step h, by time-optimal path parameterization (Pham & Pham, 2018). Each step thus runs at its target pace unless a limit binds, and then only that stretch slows down.

Splice. At t_b, P is at $\mathbf{x}_b$ and moving, not at the start $\mathbf{w}_{h_0}$ of $\boldsymbol{\xi}$, so SPLICE connects the two with a short transition. It makes two choices. *When*: SPLICE starts $\boldsymbol{\xi}$ at $t^\star$, and $\boldsymbol{\xi}(t)$ denotes its point due at time t (the ticks in Figure 2). The first kept waypoint $\mathbf{w}_{h_0}$ is thus due when P reaches step h_0, and the rest of $\boldsymbol{\xi}$ follows on its own timing. *Where*: SPLICE joins $\boldsymbol{\xi}$ at the earliest point that the arm reaches on time within its limits. Reaching $\mathbf{w}_{h_0}$ by $t^\star$ may be impossible, or may force the arm to brake hard or turn back, so later points of $\boldsymbol{\xi}$ are candidates as well. For each candidate entry $\boldsymbol{\xi}(t_e)$, Ruckig computes the fastest jerk-limited transition ψ_e from $\mathbf{s}_b$ that arrives there with the position, velocity and acceleration of $\boldsymbol{\xi}$ (Berscheid & Kröger, 2021). With D_e its duration, the transition arrives at $t_b + D_e$, and its lateness is $\ell_e = \max\big(0,\ t_b + D_e - t_e\big)$. SPLICE picks the entry with the smallest lateness and, among the entries reached on time ($\ell_e = 0$), the earliest whose transition passes within ϵ of every waypoint it skips, or the one that passes nearest if none does (Appendix F). The arm thus joins $\mathbf{w}_{h_0}$ whenever it can do so on time, and otherwise cuts the corner near it and merges further along $\boldsymbol{\xi}$. The transition followed by the rest of $\boldsymbol{\xi}$ is the spliced reference $\bar{\mathbf{x}}$, on which each waypoint $\mathbf{w}_h$ after the entry is due at a time t_h.

Smooth. Neither piece respects every limit, since the transition bounds the torque only approximately and $\boldsymbol{\xi}$ does not bound the torque rate. SMOOTH therefore solves a minimum-jerk optimal control problem on the timeline of $\bar{\mathbf{x}}$, as a quadratic program (Frison & Diehl, 2020). It starts from $\mathbf{s}_b$ at t_b, keeps $\mathbf{x}(t_h)$ within η of $\mathbf{w}_h$ for each waypoint after the entry, and enforces all limits, including the torque rate. The spliced reference only sets these times and the point about which the dynamics are linearized, and the program places no target on the transition itself. The resulting plan is checked against the limits before installation. If the program is infeasible or its plan fails the check, SMOOTH solves it again with the boxes relaxed to a penalty, and a plan that still fails is replaced by a braking plan.

5 EXPERIMENTS

5.1 TRAINING PIPELINE

Our experiments follow two stages. In both, our policy is trained on the median clock and predicts tempo channels for the 0.9, 0.95 and 0.99 quantiles (Section 3.5). In the first stage, we train the speed prior on the teleoperated demonstrations of DROID and the handheld demonstrations of FastUMI-100K, retime every DROID episode with the FastUMI-100K token, and fine-tune the released $\pi_{0.5}$ DROID policy on the retimed episodes (Khazatsky et al., 2024; Liu et al., 2025; Physical Intelligence, 2025). We evaluate this policy on RoboLab (Section 5.2). In the second stage, we adapt both models to handheld demonstrations collected with a YUBI gripper (Ohkawa et al., 2026). For each task, we fine-tune the speed prior with the shortest quarter of the episodes as the target do-



Figure 3: Real-world tasks and their instructions, each shown at its initial scene (1) and its goal (2). The policies for stacking cups, folding towel and cleaning trash are fine-tuned on handheld YUBI demonstrations, and placing bananas is performed zero-shot.

Table 1: Motion quality at matched speed-up on RoboLab. Peak is the peak end-effector speed in $\mathrm{m/s}$ and Jerk the integrated squared jerk in $10^3\,\mathrm{m^2/s^5}$. SPARC, Peak and Jerk use the episodes that every execution of the baseline and our policy in Figure 4 completes.

Execution	Speed-up	SR ↑	SPARC↑	Peak↓	Jerk↓
Uniform $1\times$	1.00	0.388	-3.99	0.26	0.41
Uniform $2\times$	1.92	0.487	-4.34	0.52	1.66
Ours, $q = 0.9$	2.05	**0.508**	$\mathbf{-3.25}$	**0.36**	**1.19**
Uniform $3\times$	2.82	0.513	-4.51	0.77	4.55
Ours, $q = 0.99$	2.76	**0.545**	$\mathbf{-3.19}$	**0.46**	**2.15**

main and retime every demonstration with the target token. We then fine-tune the first-stage policy, from its checkpoint after 20,000 updates, on the retimed demonstrations and adapt it to train-time RTC (Black et al., 2025b). We evaluate these policies on the real robot (Section 5.3).

5.2 Simulation on RoboLab

Setup. We compare our policy with two policies trained on the recorded timing of DROID. The baseline uses all episodes. Filtered BC tests whether imitating only fast demonstrations suffices and uses only the fastest episodes, 10,769 of 74,546, chosen so that their pace, $1.60\times$ that of all episodes, matches that of our median clock. All policies share the same observation frames (Appendix C). We accelerate the baseline uniformly at deployment by playing each chunk at $1.5\times$, $2\times$, $2.5\times$ or $3\times$, and play our policy on its base clock and on each tempo channel. In simulation, all policies play their chunks by interpolating the joint targets, without the controller of Section 4, so the comparison isolates the clocks (Appendix D). RoboLab is a high-fidelity simulation benchmark for analyzing task-generalist policies, with 120 new language-conditioned tasks spanning pick-and-place, stacking, rearrangement and tool use (Yang et al., 2026c). Retiming methods that locate precise segments with a proxy policy or labeled demonstrations of the task at hand were designed for a single task (Guo et al., 2025; Kim et al., 2026a), whereas here one policy learns from 74,546 episodes of many tasks and is evaluated on tasks it has not seen. We therefore compare with uniform acceleration, which applies to any policy. We run 10 episodes per task, and an episode fails if it exceeds the time limit of its task, so faster execution alone raises the success rate. We therefore compare policies at matched speed-up, the mean end-effector speed relative to the baseline at $1\times$.

Results. Uniform acceleration raises the success rate of the baseline from 38.8% at $1\times$ to 51.3% at $2.8\times$ (Figure 4). Our policy lies above this curve on all four clocks, by 3.0 percentage points on average at the same speed-up (95% CI [1.5, 4.5], paired bootstrap over tasks). Filtered BC runs only $1.04\times$ faster than the baseline, although its demonstrations are as fast as our median clock. Our policy also moves more smoothly (Table 1). At $2.8\times$, the spectral arc length (SPARC) of its end-effector speed is -3.19 against -4.51 under uniform acceleration, and its peak speed and jerk are 41% and 53% lower (Balasubramanian et al., 2015).

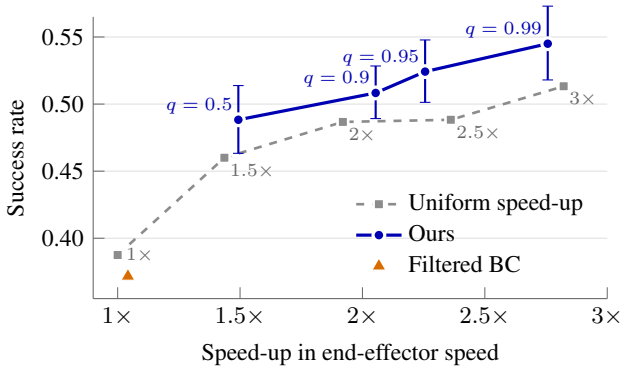

Figure 4: Success rate and speed-up on RoboLab. Gray labels are commanded speed-ups. Bars are 95% intervals of the gain over the uniform curve, bootstrapped over tasks.

5.3 Real-robot experiments

Data. We collect handheld demonstrations of three tasks with a single YUBI gripper (Ohkawa et al., 2026), 412 episodes of stacking cups, 242 of folding a towel and 61 of cleaning trash (Figure 3), and map the tracked gripper poses to joint positions of the Franka Research 3 by inverse kinematics. The robot carries the same gripper with a motor-driven jaw. The shortest quarter of the episodes of each task forms the target domain, 98, 58 and 15 episodes, which hold 16%, 16% and 20% of the frames, respectively.

Baselines and execution. We compare our policy with a default policy trained on all episodes and an expert policy trained only on the target episodes of the shortest quarter, both on their recorded timing and with train-time RTC (Black et al., 2025b). Unlike in simulation, all policies run on the robot under the controller of Section 4, at the constant pace $\alpha = 1$ and, for our policy, also on each of its tempo channels q90, q95 and q99. To measure the effect of the controller, the default policy also runs under the PD baseline of Appendix F, which holds each action as a position target for one step at $15\,\mathrm{Hz}$ as the openpi DROID example does (Physical Intelligence, 2025).

Tasks and metrics. Figure 3 shows each task with its instruction. A cup episode succeeds if the three cups form a pyramid within $10\,\mathrm{s}$, and a towel episode if the towel is folded within $10\,\mathrm{s}$. We run 30 episodes per condition on these two tasks and report the success rate (SR) together with the throughput with regret (TPR) and the average time of the successful rollouts (ATR) of Arachchige et al. (2025). TPR rewards each success by the inverse of its completion time and penalizes each failure by the inverse of the time limit, so it favors policies that are both fast and reliable. In cleaning trash, the robot has $60\,\mathrm{s}$ to put 15 trash items into the bin, and we report the throughput in units per hour (UPH), the number of items placed per hour, over 5 episodes. For all tasks, we also report SPARC averaged over all episodes. Appendix G gives the definitions.

Results. Table 2 reports the results. Under our controller, the default policy stacks the cups in 29 of 30 episodes and takes $5.67\,\mathrm{s}$ on average. Our policy takes $4.32\,\mathrm{s}$ on its base clock and between $3.15\,\mathrm{s}$ and $3.57\,\mathrm{s}$ on its tempo channels, at success rates between 0.87 and 1.00. With the 0.95 tempo channel, it stacks the cups in all 30 episodes in $3.43\,\mathrm{s}$, which raises TPR from 10.03 to 17.50. For reference, SAIL reports $14.71\,\mathrm{s}$ at a success rate of 0.40 on the same task, with a diffusion policy trained on 50 teleoperated demonstrations and accelerated to $1.76\times$ their pace, although on a different robot (Arachchige et al., 2025). Our policy thus stacks the cups $4.3\times$ faster than SAIL and succeeds in every episode. The expert policy is also fast when it succeeds, at $4.52\,\mathrm{s}$, but it learns only from the shortest quarter of the episodes and succeeds in only 4 of 30 episodes. In towel folding, our policy matches the time of the default policy on its base clock, $5.23\,\mathrm{s}$ against $5.19\,\mathrm{s}$, at a success rate of 0.97 against 0.87, and its tempo channels shorten the average time to between $3.89\,\mathrm{s}$ and $4.39\,\mathrm{s}$ at success rates between 0.90 and 0.97. In cleaning trash, our policy places 654 items per hour on its base clock and up to 862 with the 0.95 tempo channel, compared with 504 for the default policy and 564 for the expert policy.

Under the PD baseline, the same default policy stacks the cups in 22 of 30 episodes, folds the towel in 1 of 30 and places 48 trash items per hour, compared with 29, 26 and 504 under our controller. On cups and trash, our policy also moves more smoothly than the default policy on every clock, with a SPARC between -2.70 and -2.62 against -2.91 on cups and between -4.88 and -4.56 against -5.20 on trash, whereas on towel its SPARC ranges from -3.52 on its base clock to -3.19 on the 0.99 tempo channel, against -3.25.

Table 2: Real-robot results on the tasks of Figure 3. Ours (1x) runs our controller at $\alpha = 1$, and Ours (q90), (q95) and (q99) on the tempo channels of the 0.9, 0.95 and 0.99 quantiles. TPR is in successes per minute, ATR in seconds and UPH in items per hour. Appendix G defines the metrics.

		Stacking cups				Folding towel				Cleaning trash	
Model	Controller	SR↑	TPR↑	ATR↓	SPARC↑	SR↑	TPR↑	ATR↓	SPARC↑	UPH↑	SPARC↑
Default	PD	0.73	5.86	6.13	-2.97	0.03	-5.49	6.36	$\mathbf{-3.18}$	48	-5.82
Default	Ours (1x)	0.97	10.03	5.67	-2.91	0.87	9.23	5.19	-3.25	504	-5.20
Expert	Ours (1x)	0.13	-3.43	4.52	-3.63	0.60	4.68	5.09	-3.58	564	$\mathbf{-4.46}$
Ours	Ours (1x)	0.90	11.90	4.32	-2.70	**0.97**	10.91	5.23	-3.52	654	-4.88
Ours	Ours (q90)	0.87	13.75	3.57	-2.69	**0.97**	13.04	4.39	-3.49	732	-4.56
Ours	Ours (q95)	**1.00**	**17.50**	3.43	$\mathbf{-2.62}$	0.90	12.38	4.16	-3.25	**862**	-4.77
Ours	Ours (q99)	0.90	16.56	**3.15**	-2.66	0.90	**13.42**	**3.89**	-3.19	836	-4.72

5.4 Analysis of the speed prior

We analyze the speed prior of the first stage on held-out validation episodes, reading each domain with its own token unless stated otherwise. Appendix E gives the data and further results.

Pace within an episode. To slow down where demonstrators do, such as at a grasp, the prior must predict how the pace changes within a demonstration, not only how fast the demonstration is overall. We isolate these changes by subtracting each episode's mean log speed from both the recorded and the predicted speeds. The predicted median explains 87% of what remains on DROID and 82% on FastUMI-100K (Table 3).

Transfer to the handheld pace. We next examine which parts of a DROID path become faster when the prior reads it with the FastUMI-100K token. Grasps and releases speed up more than transit, by $1.62\times$ against $1.30\times$ at the median, because handheld demonstrators slow down less around gripper events than teleoperators do (Table 4). The transferred clock thus shortens the hesitation of teleoperation rather than accelerating the motion uniformly.

Smoothness of the retimed demonstrations. Finally, we examine where the smoothness of our policies in Table 1 comes from, by comparing held-out DROID episodes before and after retiming. Retiming with the median clock raises SPARC from -3.91 to -2.84, whereas uniformly accelerating each episode to the same duration leaves it at -3.85, since it keeps the stops and hesitations of the recording. The smoothness is therefore already present in the demonstrations that our policies are trained on.

6 Conclusion

We learned a speed prior that predicts how fast demonstrators traverse each part of an end-effector path, and used it to give demonstrations the pace of faster ones, those recorded with a handheld gripper or the shortest quarter of the episodes of a task. Trained once across tasks, the prior retimes demonstrations whose paths the faster demonstrators never followed. Training only on the faster demonstrations does not yield a policy that is both fast and reliable, whereas giving their pace to all demonstrations does. At the same speed-up, the resulting policies succeed more often than uniformly accelerated policies and move more smoothly, and our controller executes their pace within the limits of a real robot. One limitation is that the prior reads only the path and the instruction, so it cannot slow down for difficulty that does not show in the path, such as a fragile object.

AI USE STATEMENT

We used AI coding assistants (Claude Code and Codex) to help implement our methods: the authors designed the method, including its mathematical formulation and architecture, and the tools generated code according to our specifications. We also used generative AI tools to polish the writing of author-drafted text. We have not used generative AI tools for any other task requiring disclosure (e.g., generating data, developing theory or proofs, proposing hypotheses, designing experiments, translation, data analysis, or interpreting results). All AI-generated code was reviewed line by line and verified for correctness by the authors, and all AI-assisted edits to the text were checked to ensure they did not change the technical content. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We use publicly available datasets (e.g., DROID and FastUMI) under their respective licenses, along with handheld demonstration data collected by members of our research team that contain no personally identifiable information. All real-robot experiments were conducted in a controlled laboratory environment with appropriate safety measures. As learned policies may behave unpredictably in unseen settings, real-world deployment near humans would require additional safety validation.

REFERENCES

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

Sivakumar Balasubramanian, Alejandro Melendez-Calderon, Agnes Roby-Brami, and Etienne Burdet. On the analysis of movement smoothness. *Journal of neuroengineering and rehabilitation*, 2015.

Lars Berscheid and Torsten Kröger. Jerk-limited real-time trajectory generation with arbitrary target states. In *RSS*, 2021.

Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *NeurIPS*, 2025a.

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

Giovanni Braglia, Davide Tebaldi, André E Lazzaretti, and Luigi Biagiotti. Arc-length-based warping for robot skill synthesis from multiple demonstrations. In *IROS*, 2025.

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

Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl. In *ICML*, 2024.

Tamar Flash and Neville Hogan. The coordination of arm movements: an experimentally confirmed mathematical model. *Journal of Neuroscience*, 5(7):1688–1703, 1985.

Gianluca Frison and Moritz Diehl. HPIPM: a high-performance quadratic programming framework for model predictive control. *IFAC-PapersOnLine*, 2020.

Lingxiao Guo, Zhengrong Xue, Zijing Xu, and Huazhe Xu. Demospeedup: Accelerating visuomotor policies via entropy-guided demonstration acceleration. In *CoRL*, 2025.

Yifan Hou, Zeyi Liu, Cheng Chi, Eric Cousineau, Naveen Kuppuswamy, Siyuan Feng, Benjamin Burchfiel, and Shuran Song. Adaptive compliance policy: Learning approximate compliance for diffusion guided control. In *ICRA*, 2025.

Qingda Hu, Ziheng Qiu, Jieru Zhao, Zhongxue Gan, and Wenchao Ding. Autospeed: Annotation-free stage-adaptive motion speed learning for robot manipulation. In *ECCV*, 2026.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *CoRL*, 2025.

Dong Jing, Jingchen Nie, Tianqi Zhang, Jiaqi Liu, Huaxiu Yao, Zhiwu Lu, and Mingyu Ding. Tempovla: Learning speed-controllable vision-language-action policies. *arXiv preprint arXiv:2606.06491*, 2026.

Tatsuya Kamijo, Cristian C Beltran-Hernandez, and Masashi Hamaya. Learning variable compliance control from a few demonstrations for bimanual robot with haptic feedback teleoperation system. In *IROS*, 2024.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

Byung-ju Kim, Jinu Pahk, Chungwoo Lee, Jaejoon Kim, Jangha Lee, Theo Taeyeong Kim, Kyuhwan Shim, Jun Ki Lee, and Byoung-Tak Zhang. Espada: Execution speedup via semantics aware demonstration data downsampling for imitation learning. *IEEE Robotics and Automation Letters*, 2026a.

Jongmin Kim, Junsu Ha, Che-Sang Park, et al. CompVLA: A variable compliance vision-language-action model for contact-rich manipulation. In *CoRL*, 2026b.

Sunwoo Kim, Jeongjun Kim, and Joseph J Lim. Time optimal execution of action chunk policies beyond demonstration speed. In *ICLR*, 2026c.

Yixin Lin, Austin S. Wang, Giovanni Sutanto, Akshara Rai, and Franziska Meier. Polymetis. `https://facebookresearch.github.io/fairo/polymetis/`, 2021.

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

Roberto Martín-Martín, Michelle A Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. In *IROS*, 2019.

Taewook Nam, Junmo Cho, Youngsoo Jang, and Sung Ju Hwang. Speedaug: Policy acceleration via tempo-enriched policy and rl fine-tuning. *arXiv preprint arXiv:2512.00062*, 2025.

Takehiko Ohkawa, Jumpei Arima, Yuki Noguchi, Masatoshi Tateno, Makoto Sugiura, Takuya Okubo, Kengo Ikeuchi, Yuma Shin, Hiroki Nishizawa, Naoaki Kanazawa, et al. YUBI: Yielding universal bidigital interface for bimanual dexterous manipulation at scale. *arXiv preprint arXiv:2606.10244*, 2026.

Frank Chongwoo Park. Distance metrics on the rigid-body motions with applications to mechanism design. *Journal of Mechanical Design*, 1995.

Hung Pham and Quang-Cuong Pham. A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Transactions on Robotics*, 2018.

Physical Intelligence. openpi. `https://github.com/Physical-Intelligence/openpi`, 2025.

Robert Ramirez Sanchez, Daniel J Evans, Dylan P Losey, and Siddarth Jain. Volt: Vision and language trajectory segmentation for faster-than-demonstration policies. *arXiv preprint arXiv:2606.06323*, 2026.

Arno Verduyn, Maxim Vochten, and Joris De Schutter. Enhancing motion trajectory segmentation of rigid bodies using a novel screw-based trajectory-shape representation. In *ICRA*, 2024.

Chen Yang, Yucheng Hu, Yunchao Ma, Yunhuan Yang, Jing Tan, and Haoqiang Fan. Realtime-VLA V2: Learning to run VLAs fast, smooth, and accurate. *arXiv preprint arXiv:2603.26360*, 2026a.

Licheng Yang, Lingfeng Qian, Fei Zheng, Yonghao He, Wei Sui, Shuangshuang Li, and Hu Su. Improving robotic imitation learning via trajectory standardization. *arXiv preprint arXiv:2606.22907*, 2026b.

Xuning Yang, Rishit Dagli, Alex Zook, Hugo Hadfield, Ankit Goyal, Stan Birchfield, Fabio Ramos, and Jonathan Tremblay. Robolab: A high-fidelity simulation benchmark for analysis of task generalist policies. *arXiv preprint arXiv:2604.09860*, 2026c.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

Yongsheng Zhao, Lei Zhao, Baoping Cheng, Gongxin Yao, Xuanzhang Wen, and Han Gao. VLA-RAIL: A real-time asynchronous inference linker for VLA models and robots. *arXiv preprint arXiv:2512.24673*, 2025.

A HANDHELD DEMONSTRATIONS

Frames. We express both domains at the flange of the robot arm. DROID records this pose directly. FastUMI-100K records the pose of the handheld device, a position $\mathbf{p}$ and an orientation R decoded from extrinsic xyz Euler angles, in a local frame with $+x$ forward, $+y$ left and $+z$ up on the device. We map it to the common frame by

$$\mathbf{p}' = \mathbf{p} + R\,\mathbf{b}, \qquad R' = RA, \qquad A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \qquad \mathbf{b} = (-0.25, 0, 0)\,\mathrm{m},$$

so that the common z axis points along the approach direction and the reference point lies $25\,\mathrm{cm}$ behind the recorded pose along it, where it plays the role of the robot flange. The offset is an adopted tool geometry rather than a calibrated one. We verified the axis assignment by correlating pose changes with image motion on 24,618 pairs of adjacent frames from 72 episodes, and all 12 tasks among them selected the same right-handed assignment. Gripper readings are converted to an opening between 0 and 1 in both domains, with the closure recorded by DROID mapped to one minus its value.

Tracking jumps. Handheld poses come from visual tracking, which occasionally jumps. We remove every handheld episode that contains an interval faster than $2\,\mathrm{m/s}$ or a translational spike, an interval more than five times faster than the median of the eight intervals on each side whose rotation accounts for less than 10% of its arc length. This removes 1,358 training and 78 validation episodes. Robot episodes are not filtered, and the input normalization is fitted after the removal.

B SPEED PRIOR TRAINING

Architecture. The input sequence is a domain token, an instruction token and $\mathrm{node}_0, \mathrm{query}_0, \mathrm{node}_1, \ldots, \mathrm{query}_{15}, \mathrm{node}_{16}$, 35 tokens in total. Node tokens are linear projections of the seven node features, which are normalized by an affine transform fitted to the 1st and 99th percentiles of the training windows and then low-pass filtered. Near the ends of an episode, nodes outside the episode are filled with the nearest valid node before filtering. The instruction token is a linear projection of the frozen 1024-dimensional Qwen3-Embedding-0.6B embedding of the instruction (Zhang et al., 2025). The domain token and the query tokens are learned embeddings. The transformer has eight pre-norm layers of width 384 with six-head bidirectional attention, RMSNorm, query-key normalization, rotary position embeddings over consecutive positions and SwiGLU feed-forward layers.

Optimization. Windows of both domains are pooled and sampled uniformly without replacement, so each domain contributes in proportion to its number of windows. We train with AdamW at a constant learning rate of 10^{-4}, $\beta = (0.9, 0.95)$, weight decay 0.1 on linear weights, gradient clipping at 1.0 and a batch size of 1,024 for 30,000 updates. Every 1,000 updates, we evaluate the cross-entropy on the validation episodes, weighting episodes equally within a domain and domains equally, and keep the checkpoint with the lowest value. For each domain, 5% of the episodes are held out for validation.

Clock details. The speed of the last interval extends over any partial interval at the end of an episode. Episodes with fewer than three nodes keep their recorded timestamps.

C POLICY TRAINING

All policies start from the released $\pi_{0.5}$ checkpoint for DROID with joint-position actions (Physical Intelligence, 2025). The vision-language backbone is frozen, and only the action expert is fine-tuned. Action chunks have $H = 15$ steps at $15\,\mathrm{Hz}$. Each policy is trained for 20,000 updates with a batch size of 256 and a learning rate of 5×10^{-5} after 1,000 warm-up steps. Observation frames are selected on the recorded timing, so all policies are trained on the same observation frames regardless of their clock. The joint and gripper dimensions reuse the action normalization of the released checkpoint, and the tempo channels are normalized with statistics computed on the training chunks. A step without motion has a tempo channel of zero.

D EXECUTION IN ROBOLAB

Playback at a pace. The policy predicts $H = 15$ joint waypoints on its base clock, one per control period $\delta = 1/15\,\mathrm{s}$. On the tempo channel of quantile q, step h instead lasts $\delta \exp\left(-\tau_h^{(q)}\right)$, and under a uniform speed-up α, every step lasts δ/α. The measured joint position followed by the waypoints then forms a timed polyline, which we sample at every full control period by linear interpolation, dropping the fractional period at its end. The gripper takes the command of the step in progress.

Tracking. The joint reference interpolates the sampled waypoints linearly from the measured joint position at the $120\,\mathrm{Hz}$ physics rate, with each segment's slope as its velocity and zero acceleration, and keeps the last slope at the end of a chunk instead of stopping. A computed-torque controller tracks this reference,

$$\mathbf{u} = \mathbf{M}(\mathbf{x})\big(k_\mathrm{p}(\mathbf{x}_\mathrm{ref} - \mathbf{x}) + k_\mathrm{d}(\dot{\mathbf{x}}_\mathrm{ref} - \dot{\mathbf{x}}) + \ddot{\mathbf{x}}_\mathrm{ref}\big) + \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}})\,\dot{\mathbf{x}},$$

with $k_\mathrm{p} = 300$ and the critically damped $k_\mathrm{d} = 2\sqrt{k_\mathrm{p}}$, where $\mathbf{M}$ and $\mathbf{C}$ are read from the simulator and gravity is disabled on the arm. The torque is clamped to the effort limits of the simulated arm. The gripper command, an opening between 0 and 1, sets the position target of the fingers. Unlike the controller of Section 4, this execution neither plans within the robot's limits nor joins consecutive chunks with a transition, and each chunk starts from the measured joint position.

Replanning. The baseline at $1\times$ executes all 15 steps of each chunk before querying the policy again, which is its default setting. Accelerated chunks cover the same stretch of path in fewer control periods, so every accelerated execution, including ours, runs up to 8 control periods of each chunk before the next query. Filtered BC also runs 8 control periods of each chunk. The simulation pauses during inference, so the latency of the policy does not affect the results.

E ADDITIONAL ANALYSIS OF THE SPEED PRIOR

Accuracy. The held-out validation episodes hold 324,385 DROID windows from 3,627 episodes and 262,532 FastUMI-100K windows from 1,892 episodes, and each window contributes the speed of its center interval. Table 3 compares the predicted distributions with the recorded speeds. The predicted median explains 87% of the variance of held-out log speed on DROID and 82% on FastUMI-100K, and its pinball loss is a third of that of an input-blind predictor on DROID and two fifths on FastUMI-100K. The prior thus reads most of the pace from the path and the instruction. Its quantiles are close to their nominal levels on FastUMI-100K, where 89.0% of held-out speeds lie at or below the 0.9 quantile. On DROID the upper tail is somewhat narrow, with 84.3% at or below the 0.9 quantile and 96.7% at or below the 0.99 quantile, so the upper quantiles slightly understate how fast DROID operators can move.

Tracking jitter. Handheld poses come from visual tracking, whose jitter differs from that of robot joint encoders. In both domains, the energy of the discarded high-frequency DCT band falls with speed (Spearman correlation -0.49 on DROID and -0.56 on FastUMI-100K), because slow motion leaves relatively more jitter within a window. A prior that sees this band can therefore read the absence of tracking jitter in robot data as a cue for the fast handheld pace. The same prior trained without the low-pass filter of Section 3.3 raises the median speed of DROID by $1.80\times$ when read with the FastUMI-100K token, and adding Gaussian jitter of $0.9\,\mathrm{mm}$ and 0.35° per axis, an upper bound on the high-frequency residual of FastUMI-100K, to the DROID windows reduces this to $0.88\times$. With the filter, the same jitter changes the speed-up only from $1.35\times$ to $1.27\times$, and the residual of the prior is nearly uncorrelated with the discarded band (Spearman correlation 0.015 on DROID and -0.049 on FastUMI-100K). The transferred pace therefore mainly reflects how handheld demonstrators move rather than how they are tracked.

Retimed DROID demonstrations. Reading DROID with the FastUMI-100K token raises the predicted speed by $1.35\times$ at the median, $1.43\times$ at the 0.9 quantile and $1.55\times$ at the 0.99 quantile on average. On 2,000 held-out DROID episodes, the median clock shortens the episodes by a factor of 1.65 and the clock of the 0.99 quantile by a factor of 2.97. Their SPARC is -2.84 and -2.71, against -3.91 for the recorded motion and -3.85 and -3.70 under uniform acceleration to the same durations.

Table 3: Accuracy of the speed prior on held-out validation windows, each domain read with its own token. Coverage is the fraction of recorded speeds at or below the predicted quantile. Pinball losses are in natural log speed, for the prior and for an input-blind predictor that returns the training quantiles of the domain. R^2 is the fraction of the variance of recorded log speed explained by the predicted median, within episodes after removing each episode's mean and between episodes among the episode means, over the episodes with at least five windows. The variance between episodes is 19% of the total on DROID and 13% on FastUMI-100K, and the R^2 over all windows, 0.87 and 0.82, is the average of the two columns weighted by these shares.

	Coverage (%)				Pinball, $q = 0.5$		Pinball, $q = 0.9$		R^2 of the median	
Domain	$q = 0.1$	0.5	0.9	0.99	prior	input-blind	prior	input-blind	within	between episodes
DROID	13.0	48.1	84.3	96.7	0.087	0.261	0.037	0.089	0.87	0.87
FastUMI-100K	12.0	50.9	89.0	98.5	0.081	0.200	0.035	0.076	0.82	0.83

Table 4: Speed around gripper events on held-out validation windows. Windows are grouped by the arc length from their center interval to the nearest interval in which the gripper opening changes by more than 0.05. Speeds are medians relative to the transit group, whose recorded and predicted medians are $228\,\mathrm{mm/s}$ and $227\,\mathrm{mm/s}$ on DROID and $274\,\mathrm{mm/s}$ and $277\,\mathrm{mm/s}$ on FastUMI-100K. The median clock reads DROID with the FastUMI-100K token, and its speed-up is the geometric mean ratio of its median to the median read with the DROID token.

	DROID					FastUMI-100K		
Distance to gripper event	Share (%)	Recorded	Prior	Median clock	Speed-up	Share (%)	Recorded	Prior
Same interval	8.7	0.39	0.36	0.50	1.62	10.6	0.60	0.59
2 cm to 4 cm	9.6	0.49	0.48	0.57	1.47	10.9	0.68	0.67
6 cm to 10 cm	13.1	0.69	0.69	0.75	1.37	14.2	0.85	0.85
12 cm to 16 cm	9.7	0.84	0.84	0.89	1.33	10.9	0.99	0.99
Transit, none within 16 cm	58.9	1.00	1.00	1.00	1.30	53.4	1.00	1.00

F CONTROLLER

Algorithm 1 summarizes one cycle of the controller. Inequalities between vectors hold per joint, and $\oslash$ denotes elementwise division. $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{g}$ are the mass matrix, the Coriolis matrix and the gravity torque of the arm with its tool, $\bar{\mathbf{v}}(\mathbf{x})$ is the robot's position-dependent velocity bound, $\bar{\mathbf{a}}$, $\bar{\boldsymbol{\tau}}$ and $\dot{\bar{\boldsymbol{\tau}}}$ are the acceleration, torque and torque rate limits, and $\beta_v, \beta_a, \beta_\tau \le 1$ are planning margins on them. The acceleration scale $\tilde{\mathbf{a}}$ is $\bar{\mathbf{a}}$ where the setup has an acceleration bound and otherwise $(\bar{\boldsymbol{\tau}} - |\mathbf{g}(\mathbf{x}_b)|) \oslash \operatorname{diag} \mathbf{M}(\mathbf{x}_b)$, the acceleration that the torque limit leaves after gravity.

Timing. A worker runs inference and the solver, a $100\,\mathrm{Hz}$ client schedules observations and installs plans, and a $1\,\mathrm{kHz}$ torque executor plays the active plan, sampled every $1\,\mathrm{ms}$ with its progress p, and swaps in the new plan exactly at t_b. Each chunk runs open loop for eight steps, and the next observation is taken when P reaches the first whole step at most one pipeline latency, the sum of the 90th percentiles of t_{vla}, t_{solve} and t_{install}, before that step, so that the observation lies on the action grid of P. The switch time t_b lies the 90th percentiles of t_{solve} and t_{install} plus one client period ahead. With train-time RTC (Black et al., 2025b), the policy keeps the first d unexecuted actions of P fixed, where d is the progress of P over the 95th percentile of the time from an observation to the installation of its plan, rounded up and clipped to the delays that the policy was trained with, and t_b is delayed until P reaches progress d. These delays are $\{0, \dots, 4\}$ for our policy, whose tempo channels shorten the steps so that the same latency spans more of them, and $\{0, 1, 2\}$ for the default and expert policies. The client drops the first $\lfloor p(t_b) \rfloor$ actions, exactly the fixed ones when t_b is met as planned, and passes the consumed fraction $\phi = p(t_b) - \lfloor p(t_b) \rfloor$ of step h_0.

Target schedule. Step $h > h_0$ lasts at least $T_h^{\min} = \max\big(\delta/r_h,\ |c_h - c_{h-1}|/\mu\big)$, where μ is the gripper speed, so that the arm does not outrun the fingers, and step h_0 at least $T_{h_0}^{\min} = \max\big((1-\phi)\,\delta/r_{h_0},\ |c_{h_0} - c_b|/\mu\big)$, with c_b the gripper command at t_b. The due time $t^\star$ is likewise postponed until the gripper can reach c_{h_0}. When P ends before step h_0, as for the first chunk or after a braking plan, $t^\star = t_b + T_{h_0}^{\min}$.

Algorithm 1 Pace aware chunk execution

Input policy π, tempo channel q or speed-up α, robot limits $\mathcal{L}$, box radius η, blend tolerance ϵ

while the task runs **do** ▷ the robot tracks the running plan P
  $o \leftarrow \textsc{Observe}(P)$ ▷ at t_o; with RTC, o includes the unexecuted actions of P
  $(\mathbf{w}_h, c_h, \tau_h^{(q)})_{h=1}^{H} \leftarrow \pi(o),\ r_h \leftarrow \exp\left(\tau_h^{(q)}\right)$ or $r_h \leftarrow \alpha$ ▷ t_{vla}
  $(t_b, \mathbf{s}_b, h_0, t^\star) \leftarrow \textsc{Reserve}(P)$ ▷ $h_0 = \lfloor p(t_b) \rfloor + 1,\ p(t^\star) = h_0$
  $\boldsymbol{\xi} \leftarrow \textsc{Draft}(\mathbf{w}_{h_0:H}, c_{h_0:H}, r_{h_0:H})$ ▷ t_{solve} until $\textsc{Install}$
  $\bar{\mathbf{x}} \leftarrow \textsc{Splice}(\mathbf{s}_b, t^\star, \boldsymbol{\xi})$
  $P' \leftarrow \textsc{Smooth}(\mathbf{s}_b, \bar{\mathbf{x}})$
  if a step fails or not $\textsc{Verify}(P', \mathcal{L})$ **then**
    $P' \leftarrow \textsc{Brake}(\mathbf{s}_b)$
  $\textsc{Install}(P', t_b)$ ▷ t_{install}, then $P \leftarrow P'$ at t_b

function $\textsc{Draft}(\mathbf{w}, c, r)$
  $\boldsymbol{\gamma} \leftarrow \textsc{MinJerkPath}(\mathbf{w}, \eta)$ ▷ geometry on the base clock σ
  return $\textsc{TimeOptimal}(\boldsymbol{\gamma}, \mathcal{L}, T^{\min}(c, r))$ ▷ timing, each step no shorter than $T_h^{\min}$

function $\textsc{Splice}(\mathbf{s}_b, t^\star, \boldsymbol{\xi})$
  start $\boldsymbol{\xi}$ at $t^\star$ ▷ $\boldsymbol{\xi}(t)$ is due at t
  for each entry time $t_e \geq t^\star$ **do** ▷ $\boldsymbol{\xi}(t^\star) = \mathbf{w}_{h_0}$
    $\boldsymbol{\psi}_e \leftarrow \textsc{Ruckig}(\mathbf{s}_b, \boldsymbol{\xi}(t_e), \mathcal{L})$ ▷ fastest transition, duration D_e
    $\ell_e \leftarrow \max\left(0,\ t_b + D_e - t_e\right)$ ▷ lateness
  $e^\star \leftarrow \arg\operatorname{lexmin}_e\left(\ell_e, b_e, e\right)$ ▷ b_e: excess over ϵ at the skipped waypoints
  return $\boldsymbol{\psi}_{e^\star} \oplus \boldsymbol{\xi}|_{t \geq t_{e^\star}}$ ▷ $\oplus$ concatenates in time; $\boldsymbol{\psi}$ to $\mathbf{w}_{h_0}$ ends no earlier than $t^\star$

function $\textsc{Smooth}(\mathbf{s}_b, \bar{\mathbf{x}})$
  $P' \leftarrow \textsc{JerkQP}(\mathbf{s}_b, \bar{\mathbf{x}}, \eta, \mathcal{L})$ ▷ boxes at the due times t_h on $\bar{\mathbf{x}}$
  if P' is infeasible or not $\textsc{Verify}(P', \mathcal{L})$ **then**
    $P' \leftarrow$ the same program with the boxes as a penalty
  return P'

DRAFT. The path $\boldsymbol{\gamma}$ is piecewise quadratic in σ, with the acceleration $\mathbf{a}_m$ constant over rows of $32\,\text{ms}$. It starts at $\mathbf{w}_{h_0}$ with the secant velocity toward $\mathbf{w}_{h_0+1}$ and minimizes $\sum_m \|(\mathbf{a}_m - \mathbf{a}_{m-1}) \oslash \tilde{\mathbf{a}}\|^2$ subject to $\|\boldsymbol{\gamma}(h\delta) - \mathbf{w}_h\|_\infty \leq \eta = 5\,\text{mrad}$ and the joint range. This quadratic program has no velocity, acceleration or torque rows, so the polyline through the waypoints is feasible, and we solve it with HPIPM (Frison & Diehl, 2020) and fall back to that polyline if the solver fails. We fit a cubic spline to $\boldsymbol{\gamma}$ with knots every $24\,\text{ms}$ of σ and retime it with TOPP-RA (Pham & Pham, 2018) on 64 gridpoints plus the knots, under the scaled velocity, acceleration and torque limits, the torque from the full inverse dynamics, and under $\dot{\sigma} \leq 1.01\,\bar{r}_\nu$ on each knot span ν, where $\bar{r}_\nu$ is the length of the span divided by its minimum duration under $T_h^{\min}$. The entry rate is 0.99 times the largest feasible one, and the terminal rate is not forced to zero, so a plan ends moving and the next plan takes over before its end.

SPLICE. Candidate entries are spaced $16\,\text{ms}$ apart from t_b up to step 8, with $\boldsymbol{\xi}(t_e) = \mathbf{w}_{h_0}$ for $t_e \leq t^\star$. A transition to $\mathbf{w}_{h_0}$ is stretched so that it arrives no earlier than $t^\star$, unless it would then turn back, and a postponement of $t^\star$ for the gripper slows the first span of $\boldsymbol{\xi}$ instead. Ruckig (Berscheid & Kröger, 2021) bounds each transition by $\beta_v \bar{\mathbf{v}}$, by $\beta_a \bar{\mathbf{a}}$, or $\tilde{\mathbf{a}}$ without an acceleration bound, and by the jerk $\beta_u \bar{\mathbf{u}} \oslash \operatorname{diag} \mathbf{M}(\mathbf{x}_b)$ that the torque rate limit allows. The selected entry is

$$e^\star = \arg\operatorname*{lexmin}_e \left(\ell_e,\, b_e,\, e\right), \qquad b_e = \max\left(0,\, \max_{h \in \mathcal{S}_e} \min_t \|\boldsymbol{\psi}_e(t) - \mathbf{w}_h\|_\infty / \epsilon - 1\right),$$

where $\mathcal{S}_e$ holds the waypoints that the entry skips and $\epsilon = 0.05\,\text{rad}$. The rest of $\boldsymbol{\xi}$ follows the transition without a gap, and the gripper commands of skipped waypoints are issued by the end of the transition.

SMOOTH. The quadratic program holds the jerk $\boldsymbol{\zeta}_i$ constant over rows of $T_\text{r} = 16\,\text{ms}$, starts from $\mathbf{s}_b$ and minimizes $\sum_i \|T_\text{r}\, \boldsymbol{\zeta}_i \oslash \tilde{\mathbf{a}}\|^2$ without a tracking term. At the end of every row it enforces the joint range, the scaled velocity and acceleration limits, the scaled torque limit with the dynamics

Table 5: Settings that differ between the two setups.

Setting	YUBI	DROID zero-shot
Gripper	Yubi DYNAMIXEL	Robotiq
Real-time chunking	train-time	inference-time
Trained RTC delays d	$\{0, \dots, 4\}$ for ours, $\{0, 1, 2\}$ otherwise	–
RTC maximum guidance weight	–	5
Executed tempo quantiles	0.9, 0.95 and 0.99	0.9

linearized along $\bar{\mathbf{x}}$, and the torque rate limit $|\mathbf{M}(\bar{\mathbf{x}}_i)\,\boldsymbol{\zeta}_i| \le \beta_u \bar{\dot{\mathbf{u}}}$. Each waypoint after the entry must lie within η at t_h, and the terminal velocity within $0.05\,\mathrm{rad/s}$ of that of $\boldsymbol{\xi}$. If the program is infeasible or its plan fails verification, the boxes are replaced by a linear penalty of 10^3 per radian.

Verification and fallbacks. Each plan is checked against the unscaled limits, the velocity, acceleration and joint range every $1\,\mathrm{ms}$ and the torque from the full inverse dynamics and its rate every $8\,\mathrm{ms}$, and is accepted if it exceeds no limit by more than 5% and no joint range by more than $1\,\mathrm{mrad}$. Otherwise, and also when TOPP-RA cannot time the path or Ruckig finds no transition, the solver returns a braking plan, the program of SMOOTH without boxes and ending at rest. Its horizon is Ruckig's time-optimal stopping time rounded up to whole rows and at least $48\,\mathrm{ms}$, and is doubled up to twice when the program is infeasible or its plan fails verification. The solver refuses a chunk only when every braking plan fails or the solve takes longer than $1\,\mathrm{s}$. After a refusal or a plan that arrives after t_b, the client keeps P and solves the remaining actions again from a new switch time, at least $33\,\mathrm{ms}$ after the previous request, and once P ends it drops the chunk and takes a new observation. When P is about to end in motion with no plan pending, the client requests a braking plan itself, and past the end of any plan the executor brakes every joint at $10\,\mathrm{rad/s^2}$.

Low-level execution. The executor applies at $1\,\mathrm{kHz}$ the joint torque

$$\mathbf{u} = \big(\mathbf{J}^\top \mathbf{K}_\mathrm{p}^\mathrm{c} \mathbf{J} + \mathbf{K}_\mathrm{p}\big)(\mathbf{x}_\mathrm{ref} - \mathbf{x}) + \big(\mathbf{J}^\top \mathbf{K}_\mathrm{d}^\mathrm{c} \mathbf{J} + \mathbf{K}_\mathrm{d}\big)(\dot{\mathbf{x}}_\mathrm{ref} - \dot{\mathbf{x}}) + \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}})\,\dot{\mathbf{x}} + \kappa\,\mathbf{M}(\mathbf{x})\,\ddot{\mathbf{x}}_\mathrm{ref},$$

clamped to the torque limits, where $(\mathbf{x}_\mathrm{ref}, \dot{\mathbf{x}}_\mathrm{ref}, \ddot{\mathbf{x}}_\mathrm{ref})$ is the current plan sample, $\mathbf{J}$ is the end-effector Jacobian, $\mathbf{K}_\mathrm{p}, \mathbf{K}_\mathrm{d}$ and $\mathbf{K}_\mathrm{p}^\mathrm{c}, \mathbf{K}_\mathrm{d}^\mathrm{c}$ are the joint and Cartesian stiffness and damping gains, and the robot compensates gravity. The default gains of DROID's Polymetis configuration (Khazatsky et al., 2024; Lin et al., 2021) are $\mathbf{K}_\mathrm{p} = \mathrm{diag}(40, 30, 50, 25, 35, 25, 10)$, $\mathbf{K}_\mathrm{d} = \mathrm{diag}(4, 6, 5, 5, 3, 2, 1)$, $\mathbf{K}_\mathrm{p}^\mathrm{c} = \mathrm{diag}(400, 400, 400, 15, 15, 15)$ and $\mathbf{K}_\mathrm{d}^\mathrm{c} = \mathrm{diag}(37, 37, 37, 2, 2, 2)$ in SI units. Our controller scales the stiffness gains by 4 and the damping gains by 2 and uses $\kappa = 1$. Gripper commands are sent at $30\,\mathrm{Hz}$.

PD baseline. Following the openpi DROID example (Khazatsky et al., 2024; Physical Intelligence, 2025), the baseline holds each waypoint as a position target for $T_h^{\min}$, a $15\,\mathrm{Hz}$ stepped reference at $\alpha = 1$, and caps each new target at $0.2\,\mathrm{rad}$ from the measured position, as DROID's joint velocity interface does. It uses the torque law above with the default gains, $\dot{\mathbf{x}}_\mathrm{ref} = \ddot{\mathbf{x}}_\mathrm{ref} = \mathbf{0}$ and $\kappa = 0$, the hybrid joint impedance controller of Polymetis (Lin et al., 2021), and shares our observation timing, switch times and RTC. We keep the default gains and the cap because the stepped reference is unsafe otherwise. Without the cap it triggered the robot's safety reflex in four of five trash cleaning episodes, and with our stiffer gains it did so even at half the pace.

Settings. Table 5 lists the settings that differ between the two setups, and Table 6 those of the two controllers. Our controller plans within the firmware limits of the Franka Research 3, with velocity maxima $(2.62, 2.62, 2.62, 2.62, 5.26, 4.18, 5.26)\,\mathrm{rad/s}$, torque limits $(87, 87, 87, 87, 12, 12, 12)\,\mathrm{Nm}$ and a torque rate limit of $1000\,\mathrm{Nm/s}$. Its torque interface has no acceleration limit, so on YUBI we add $20\,\mathrm{rad/s^2}$, above which the tracking error of the wrist doubled. On YUBI, inference and the solve round trip each take $28\,\mathrm{ms}$ at the median, and $d = 2$ in most chunks.

G REAL-ROBOT EVALUATION

Each condition runs 30 episodes on stacking cups and folding towel and 5 episodes on cleaning trash. The time limit t_max is $10\,\mathrm{s}$ for cups and towel and $60\,\mathrm{s}$ for trash. The completion time t_i of

Table 6: Settings of our controller (Ours) and the PD baseline (PD), shared by both setups. An entry that spans both columns applies to each of them.

Setting	Ours	PD
Chunk and open-loop horizon	$H = 15$ actions, $\delta = 1/15\,\mathrm{s}$, 8 steps	
Stiffness and damping gains, feedforward κ	$4\times, 2\times, 1$	$1\times, 1\times, 0$
Step cap	none	$0.2\,\mathrm{rad}$

a successful episode i is read from the measured gripper. It is the time at which the second release of the gripper has progressed by 90% in stacking cups, and the time at which its last release has progressed by 90% in folding towel.

Success rate and time. SR is the fraction of successful episodes, and ATR is the mean of t_i over the successful episodes.

Throughput with regret. TPR (Arachchige et al., 2025) rewards each success by the inverse of its completion time and penalizes each failure by the inverse of the time limit. We report it in successes per minute, 60 times the per-second definition of Arachchige et al. (2025),

$$\mathrm{TPR} = \frac{60}{N} \sum_{i=1}^{N} \Big(\frac{S_i}{t_i} - \frac{1 - S_i}{t_{\max}} \Big),$$

where N is the number of episodes, $S_i = 1$ if episode i succeeds and $S_i = 0$ otherwise, and times are in seconds.

Throughput. In cleaning trash, episode i places K_i of the 15 trash items in the bin, and we define its duration T_i as the time at which the last item is placed if all 15 are placed, and $t_{\max} = 60\,\mathrm{s}$ otherwise. We report throughput in items per hour (UPH), computed over all N episodes as

$$\mathrm{UPH} = 3600\, \frac{\sum_{i=1}^{N} K_i}{\sum_{i=1}^{N} T_i}.$$

Smoothness. SPARC measures the smoothness of the end-effector speed (Balasubramanian et al., 2015), where higher values are smoother. We resample the measured end-effector position of each episode at 15 Hz, take its speed $v(t) = \|\dot{\mathbf{p}}(t)\|$ by central differences and compute the arc length of the normalized magnitude spectrum $A(f)$ of the speed,

$$\mathrm{SPARC} = -\sum_{f_k \le f_*} \sqrt{\Big(\frac{\Delta f_k}{f_*} \Big)^2 + \big(\Delta A_k \big)^2},$$

where the cutoff f_* is set by an amplitude threshold of 0.05 and a maximum of 10 Hz, which the 15 Hz sampling limits to its Nyquist frequency of 7.5 Hz. A successful episode is evaluated up to its completion and a failed one up to the end of its recording. We report the mean over all episodes on which SPARC can be computed, including the failed ones.

H Zero-shot evaluation

Setup. We evaluate the policies of Section 5.2, which are trained only on DROID, on the Franka Research 3 of Section 5.3 fitted with the Robotiq 2F-85 gripper of DROID, without further fine-tuning. The robot has 30 s to put two bananas into a tray (Figure 3). Default and Expert are the baseline and Filtered BC, trained on the recorded timing of all DROID episodes and of the fastest ones, and run under our controller at $\alpha = 1$. Our policy runs under our controller on its 0.9 tempo channel. Table 5 lists the other settings.

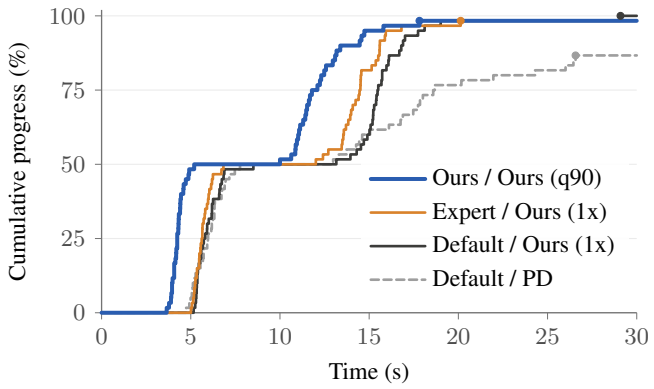

Figure 5: Mean task progress over time on the real-robot placing bananas task (30 trials per policy). Each banana placed in the tray within the 30 s limit adds 50%; curves average over trials, so the final height is the mean score and curves further up and to the left progress faster.

Metrics. We run 30 episodes per policy and rerun an episode when the operator makes an error, such as missing a mark. The operator marks each banana with a key press as it lands in the tray. An episode succeeds if it places both bananas within $t_{\max} = 30\,\mathrm{s}$. UPH pools the $N = 30$ episodes of a policy,

$$\mathrm{UPH} = \frac{3600 \sum_{i=1}^{N} K_i}{\sum_{i=1}^{N} T_i},$$

where K_i is the number of bananas that episode i places within $t_{\max}$, and T_i is the time at which it places the second one if $K_i = 2$ and $t_{\max}$ otherwise. SPARC is computed as in Appendix G.

Results. Table 7 reports the results. The UPH of our policy, 546.0, is the highest, 17% above the 468.3 of the expert policy. The motion of our policy is the smoothest, with a SPARC of -3.79 against -4.20 for the default policy and the -3.99 of the expert policy.

Table 7: Real-robot results on the placing bananas task ("put bananas into the tray"), with 30 trials per setting. PD is the baseline controller of Appendix F, Ours (1x) runs our controller at $\alpha = 1$ and Ours (q90) on the tempo channel of the 0.9 quantile. An episode succeeds if it places both bananas within 30 s.

		Placing bananas	
Model	Controller	UPH↑	SPARC↑
Default	PD	291.6	-4.82
Default	Ours (1x)	445.4	-4.20
Expert	Ours (1x)	468.3	-3.99
Ours	Ours (q90)	**546.0**	$\mathbf{-3.79}$