

RETIMING ROBOT DEMONSTRATIONS WITH A LEARNED SPEED PRIOR

Seungyong Moon* Deokjae Lee* Hansang Lee* Munkhtselmeg Battengel Youngin Kim
Seonghyeon Yun Sungjin Jeon SongEun Kim Jeongmin Park Hyun Oh Song†
Seoul National University

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, a motion path and the images along it, 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 3.50s, $1.72\times$ faster than the default policy’s 6.02s while succeeding as often, in 29 of 30 trials. Videos and further material are available at https://mllab.snu.ac.kr/projects/learned_speed_prior.

1 INTRODUCTION

The pace of robot demonstrations depends on both the collection setup and the operator. Moving a robot through a teleoperation interface is slower than moving a handheld gripper directly (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

*Equal contribution.

†Corresponding author.

path, the wrist-camera images along it 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 label that marks the kind of each demonstration, so reading a path with the label 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 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 velocity, acceleration or torque limits of the robot. Joint PD control, commonly used to execute such policies, does not plan within these limits, so fast targets can cause tracking lag or 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 give DROID 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 $2.4\times$ it does so with 52% less 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 $1.5\times$ that of all episodes, runs just $1.07\times$ 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 3.50s instead of 6.02s and succeeds in 29 of 30 trials, whereas a policy trained only on this quarter succeeds in 9 of 30 trials. On the same three-cup pyramid, SAIL reports 14.71s 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 $4.2\times$ faster and succeeds more than twice as often. 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 97% of trials instead of 13%.

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 speed to be chosen afterward (Braglia et al., 2025). ISR keeps a subset of the frames, spaced at roughly equal information, measured by velocity and acceleration (Yang et al., 2026b). Others 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 a seed demonstration with vision-language and language models and propagates the labels to the other demonstrations of the task (Kim et al., 2026a). VOLT prompts a vision-language model with examples from other tasks to label every demonstration (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.

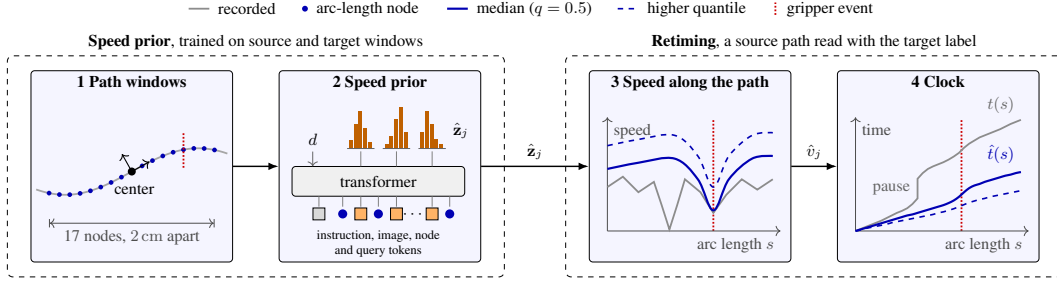


Figure 1: Learning the pace from demonstrations. (1) Paths are resampled every 2 cm of arc length and read in windows of 17 nodes. (2) The prior predicts a distribution over the log speed of every interval. (3) Reading a source path with the target label gives a quantile of the speed along the path.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

Executing policies faster. Learned policies usually send 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 the demonstrations, and scales the duration of each step 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 velocity and torque limits of the robot 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 controller parameters, 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). Each demonstration records end-effector positions $\mathbf{p}_k \in \mathbb{R}^3$, orientations $R_k \in SO(3)$ and gripper openings o_k at timestamps t_k , for $k = 0, \dots, K$. We measure progress along the path with an increment that combines translation, rotation and gripper motion,

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

where $\theta(\cdot)$ is the angle of a relative rotation and ρ_r and ρ_g convert radians and gripper openings to meters (Park, 1995). We set $\rho_r = 0.3 \text{ m/rad}$, following prior work on rigid-body trajectory segmentation (Verduyn et al., 2024), and $\rho_g = 0.15 \text{ m}$ per unit of opening. The arc length $s_k = \sum_{i < k} \Delta s_i$ grows only while the end effector or the gripper moves, so pauses do not advance it. The recorded timing is then a map $t(s)$ from arc length to time, with speed $v(s) = ds/dt$ 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 retimed to follow the pace that target demonstrations show. The two may come from different setups, such as 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 of its demonstrators move at the desired pace. Each demonstration carries a domain label $d \in \{\text{src}, \text{tgt}\}$. Handheld poses are expressed in the frame of the robot data, and episodes whose tracking fails are removed (Appendix A).

Path windows. We resample each path at nodes spaced $\Delta = 2$ 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 cm. Each node holds its position and rotation vector in the frame of the center node, its absolute gripper opening and the nearest wrist-camera image.

Speed targets. The 17 nodes bound 16 intervals. Over interval j , the arm and the gripper travel L_j^a and L_j^g , respectively (Appendix B). The target log speed of the interval is $y_j = \log(L_j/\Delta t_j)$, where $L_j = \max(L_j^a, L_j^g)$ and Δt_j is the time between its two nodes, including any pauses.

3.3 SPEED PRIOR

Architecture. The prior is a transformer trained from scratch and conditioned on the domain label. It reads an instruction token and 17 pairs of image and node tokens interleaved with 16 query tokens. Frozen encoders embed instructions and images (Zhang et al., 2025; Siméoni et al., 2025). Query tokens predict the arm speed v_j^a and the gripper speed v_j^g of their 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, not a single value, from which retiming can select a quantile. Per actuator, the prior predicts a categorical distribution over $B = 256$ equal bins of log speed from 1 to 5000 mm/s. An interval lasts as long as its slower actuator,

$$T_j = \max(L_j^a/v_j^a, L_j^g/v_j^g),$$

and treating the two speeds as independent gives the distribution $\hat{\mathbf{z}}_j$ of $\log(L_j/T_j)$ on the same bins (Appendix B). The target y_j is encoded as a two-hot vector $\mathbf{z}_j$ (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 of the window that lie within the episode.

3.4 RETIMING

To retime a source demonstration, we ask the prior how fast a target demonstrator would traverse the same path. Since the images reveal the domain, switching the label alone does not pose this question cleanly. We therefore drop the images of a window with probability 0.5 in training. Without images, the two labels differ only in the pace of the demonstrator, and we add this difference to the source reading with images, inspired by classifier-free guidance (Ho & Salimans, 2022). Writing $f^a(d, I)$ for the arm logits read under the label d with images I and $f^a(d, \emptyset)$ for those without, the arm logits for retiming are

$$\tilde{f}^a = f^a(\text{src}, I) + \lambda [f^a(\text{tgt}, \emptyset) - f^a(\text{src}, \emptyset)],$$

with a guidance coefficient λ (Appendix B). When the two setups use different grippers, the gripper keeps the source label, since its own mechanism, not the demonstrator, sets its speed. We average the distributions $\hat{\mathbf{z}}_j^{(w)}$ that the windows w covering interval j predict from these logits and decode the quantile of level q , which gives the duration $\hat{T}_j$ and the speed $\hat{v}_j = \Delta/\hat{T}_j$. The clock assigns each recorded frame the time needed to reach its arc length at these speeds,

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

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 $\pi_{0.5}$ (Intelligence et al., 2025) on chunks of H joint-position actions from the source demonstrations retimed with the median clock ($q = 0.5$), which serves as the base clock. Observations are taken at the recorded frames, and each chunk lists the joint positions and gripper commands that the retimed demonstration reaches over the next H periods of the base clock.

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 $\tau_h^{(q)} = \log(\delta/\delta_h^{(q)})$, where the step lasts $\delta = 1/15$ s on the base clock and $\delta_h^{(q)}$ on the clock of quantile q , and the policy predicts these channels as additional action dimensions (Appendix C).

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 $h = 1, \dots, H$ after the observation. The target pace $r_h = \exp(\tau_h^{(q)})$, or $r_h = \alpha$ for a constant speed-up α , bounds the duration of step h from below by δ/r_h .

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^* is the time at which P reaches that step, $p(t^*) = h_0$ (Appendix F covers a P that ends before). The solver receives the kept waypoints $\mathbf{w}_{h_0}, \dots, \mathbf{w}_H$, their target pace, $\mathbf{s}_b$, t^* and the robot limits, and returns a new plan P' . The robot switches to it at t_b ,

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

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, with no 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 ξ from the kept waypoints of the new chunk alone. It first fits a minimum-jerk path $\gamma(\sigma)$ 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

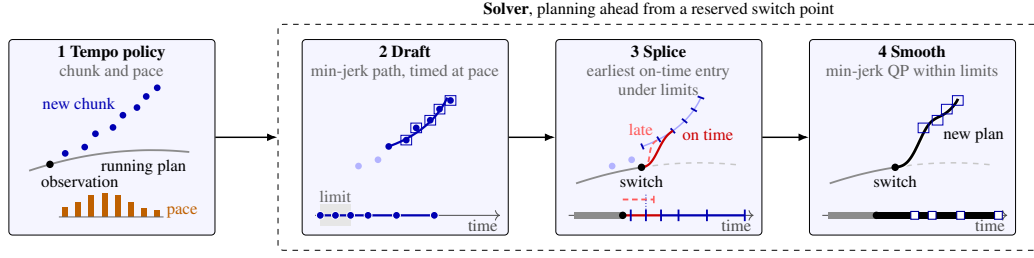


Figure 2: One execution cycle. (1) From an observation on the running plan, the policy predicts a new chunk and its pace. (2) DRAFT fits a smooth path within boxes around the kept waypoints and times it at the pace unless a limit binds. (3) SPLICE joins the running plan to this path at the earliest point 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.

its shape 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 ξ , so SPLICE connects the two with a short transition. It makes two choices. *When:* SPLICE starts ξ at t^* , and $\xi(t)$ 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 ξ follows on its own timing. *Where:* SPLICE joins ξ at the earliest point that the arm reaches on time within its limits. Reaching $\mathbf{w}_{h_0}$ by t^* may be impossible, or may force the arm to brake hard or turn back, so later points of ξ are candidates as well. For each candidate entry $\xi(t_e)$, Ruckig computes the fastest jerk-limited transition ψ_e from $\mathbf{s}_b$ that arrives there with the position, velocity and acceleration of ξ (Berscheid & Kröger, 2021). With D_e its duration, the transition arrives at $t_b + D_e$, and its lateness is $\ell_e = \max(0, t_b + D_e - t_e)$. 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 ξ . The transition followed by the rest of ξ 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 all limits, as the transition bounds the torque only roughly and ξ does not bound the torque rate. SMOOTH thus 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 have two stages. In both, our policy learns the median clock and the tempo channels of the 0.9, 0.95 and 0.99 quantiles (Section 3.5), and every policy uses train-time RTC (Black et al., 2025b). In the first stage, we train the speed prior on the handheld demonstrations of FastUMI-100K and the teleoperated ones of DROID, retime all DROID episodes with the FastUMI-100K label, and fine-tune the released $\pi_{0.5}$ DROID policy on them (Khazatsky et al., 2024; Liu et al., 2025; Physical Intelligence, 2025). We evaluate this policy in simulation on RoboLab (Section 5.2) and zero-shot on the real robot (Section 5.3). 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 episodes as the target domain and retime all demonstrations with the



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 and placing cubes are performed zero-shot (Appendix H).

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

Execution	Speed-up	SR $\uparrow$	TPR $\uparrow$	SPARC $\uparrow$	Peak $\downarrow$	Jerk $\downarrow$
Uniform $1\times$	1.00	0.462	1.45	-3.62	0.30	0.31
Uniform $2\times$	1.90	0.533	3.55	-3.87	0.52	1.23
Ours, $q = 0.95$	2.00	0.577	4.39	-3.14	0.44	0.83
Uniform $2.5\times$	2.48	0.530	3.86	-4.11	0.69	2.41
Ours, $q = 0.99$	2.38	0.608	5.16	-3.26	0.52	1.16

target label. We then fine-tune the first-stage policy on the retimed demonstrations and evaluate it on the real robot (Section 5.3).

5.2 SIMULATION ON ROBOLAB

Tasks and metrics. RoboLab (Yang et al., 2026c) is a high-fidelity benchmark for task-generalist policies in simulation. Its 120 new language-conditioned tasks span stacking, pick-and-place, tool use and rearrangement, and we run 10 episodes per task. An episode fails if it exceeds the time limit of its task, so running faster alone can raise the success rate. We therefore compare policies at matched speed-up, the mean end-effector speed relative to the default policy at $1\times$. We report the success rate (SR) and the throughput with regret (TPR) of Arachchige et al. (2025), which favors policies that are both fast and reliable (Appendix G).

Baselines. We compare our policy with two policies trained on the recorded timing of DROID. The default policy is trained on all episodes. Filtered behavior cloning (Filtered BC) is trained only on the fastest 16,609 of 74,546 episodes, whose pace of $1.5\times$ matches our median clock, and tests whether imitating fast demonstrations suffices. Methods that find the segments to accelerate need a proxy policy or labeled demonstrations of the task at hand (Guo et al., 2025; Kim et al., 2026a), whereas our policy is trained on many tasks and evaluated on unseen ones. We therefore compare with uniform acceleration, which applies to any policy: the default policy plays at $1.5\times$, $2\times$ or $2.5\times$, and 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).

Results. Uniform acceleration raises the SR of the default policy from 46.2% at $1\times$ to 53.3% at $1.9\times$ but no further (Figure 4). Our policy lies above this curve on all four clocks, on average by 4.9 percentage points in SR and 0.88 successes per minute in TPR (95% CIs [2.8, 7.1] and [0.63, 1.15]). Filtered BC runs only $1.07\times$ faster than the default policy and succeeds no more often, although its demonstrations match our median clock. At $2.4\times$, our policy moves more smoothly than uniform

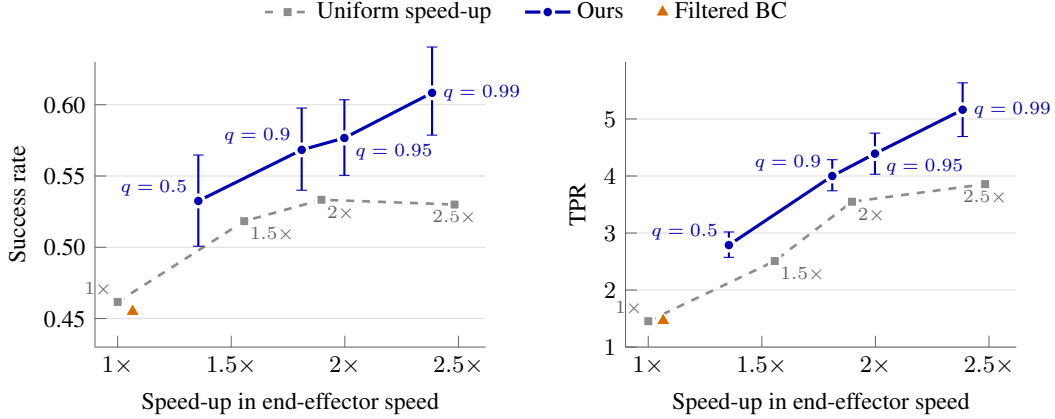


Figure 4: Success rate (left) and TPR (right) against speed-up on RoboLab. Gray labels give the commanded speed-up. TPR (Appendix G) is in successes per minute, with the time limit of each task. Bars are 95% intervals of the gain over the uniform curve, bootstrapped over tasks.

acceleration to $2.5\times$ (Table 1): the spectral arc length (SPARC) of its end-effector speed is -3.26 against -4.11 , and its peak speed and jerk are 24% and 52% lower (Balasubramanian et al., 2015).

5.3 REAL-ROBOT EXPERIMENTS

Tasks and metrics. With one YUBI gripper (Ohkawa et al., 2026), we collect demonstrations of three tasks by hand: 412 episodes of stacking cups, 242 of folding a towel and 61 of cleaning trash (Figure 3). The tracked gripper poses are mapped by inverse kinematics to joint positions of the Franka Research 3, which carries the same gripper. The shortest quarter of the episodes of each task forms the target domain, with 98, 58 and 15 episodes and 16%, 16% and 20% of the frames. A cup or towel episode succeeds if the three cups form a pyramid or the towel is folded within 10 s. On these two tasks, we run 30 episodes per condition and report SR, the average time of the successful rollouts (ATR), TPR and SPARC (Appendix G). In cleaning trash, the robot has 60 s to put 15 items into the bin, and we report the items placed per hour (UPH) and SPARC over 5 episodes.

Baselines. We compare our policy with the default policy, trained on all episodes, and Filtered BC, trained only on the target episodes, both on their recorded timing and fine-tuned from the same first-stage checkpoint as our policy. Unlike in simulation, all policies run under the controller of Section 4: the baselines at $\alpha = 1$, and our policy at $\alpha = 1$ and on each of its tempo channels q_{90} , q_{95} and q_{99} . 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 Hz, as the openpi DROID example does (Physical Intelligence, 2025).

Results. Table 2 reports the results. In stacking cups, the default policy takes 6.02 s. Our policy takes 4.40 s on its base clock and between 3.25 s and 3.65 s on its tempo channels, at success rates of 0.93 to 0.97. On the 0.95 tempo channel, it succeeds as often as the default policy, in 29 of 30 episodes, but takes only 3.50 s, raising TPR from 9.45 to 16.38. For reference, SAIL reports 14.71 s at a success rate of 0.40 on the same task, accelerating a diffusion policy trained on 50 teleoperated demonstrations to $1.76\times$ their pace, although on a different robot (Arachchige et al., 2025). Our policy is thus $4.2\times$ faster and succeeds more than twice as often. Filtered BC succeeds in only 9 of 30 episodes. In folding towel, the tempo channels shorten the time from 5.52 s to between 3.85 s and 4.39 s at success rates of 0.97 to 1.00. In cleaning trash, they raise UPH from 444 to up to 809.

Under the PD baseline, the default policy succeeds in 28 and only 4 of 30 episodes on cups and towel and places no trash items, against 29, 29 and 444 items per hour under our controller. On towel, our policy also moves more smoothly than the default policy on every clock, with a SPARC between -3.29 and -3.04 against -3.75 . On cups and trash, its SPARC stays close to that of the default policy, between -3.00 and -2.70 against -2.68 and between -5.17 and -4.49 against -4.82 .

Zero-shot transfer. Without fine-tuning, we also run the default and our policies of Section 5.2 on this robot with the Robotiq gripper of DROID, on placing bananas and placing cubes (Figure 3). On its 0.99 tempo channel, our policy places both bananas in all 30 episodes and all nine cubes in 12, whereas the default policy under our controller does so in 22 and none. UPH rises from 232 to 493 on bananas and from 354 to 511 on cubes (Appendix H). Under the PD baseline, the default policy places both bananas in only 17 episodes and 3.90 cubes on average, and our policy moves most smoothly on its base clock in both tasks.

Table 2: Real-robot results on the YUBI 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.

Model	Controller	Stacking cups				Folding towel				Cleaning trash	
		SR $\uparrow$	TPR $\uparrow$	ATR $\downarrow$	SPARC $\uparrow$	SR $\uparrow$	TPR $\uparrow$	ATR $\downarrow$	SPARC $\uparrow$	UPH $\uparrow$	SPARC $\uparrow$
Default	PD	0.93	8.25	6.48	-2.79	0.13	-4.02	6.81	-3.46	0	-5.41
Default	Ours (1x)	0.97	9.45	6.02	-2.68	0.97	10.31	5.52	-3.75	444	-4.82
Filtered BC	Ours (1x)	0.30	-1.59	7.14	-3.80	0.60	4.25	5.41	-3.46	348	-4.35
Ours	Ours (1x)	0.97	12.99	4.40	-2.70	0.97	10.59	5.38	-3.29	600	-4.49
Ours	Ours (q90)	0.97	15.68	3.65	-2.72	0.97	13.01	4.39	-3.19	732	-5.17
Ours	Ours (q95)	0.97	16.38	3.50	-2.82	0.97	13.62	4.20	-3.14	792	-4.85
Ours	Ours (q99)	0.93	16.84	3.25	-3.00	1.00	15.61	3.85	-3.04	809	-4.84

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 label 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 94% of what remains on DROID and 92% 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 label. The arm speeds up more next to grasps and releases than in transit, by $1.57\times$ against $1.28\times$ 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 -3.09 , whereas uniform acceleration to the same duration leaves it at -3.87 , 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 only transfers paces that some demonstrators have shown, so it cannot speed up motion beyond what is observed in the demonstrations.

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

- Yifan Hou, Zeyi Liu, Cheng Chi, Eric Cousineau, Naveen Kuppaswamy, 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 Taeyeon 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.
- Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.
- 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.
- Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.
- 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

This appendix concerns FastUMI-100K, the handheld data of the first stage. In the second stage, inverse kinematics maps the YUBI demonstrations to the robot instead (Section 5.3).

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

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

so that the common z axis points along the approach direction and the reference point lies 25 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 drops out or jumps. We remove every FastUMI-100K episode with a dropout, two consecutive frames with identical poses, or a jump, a change of the arm speed by more than 500 mm/s between consecutive frames. This removes 2,862 of 39,403 episodes. Robot episodes are not filtered, and the input normalization is fitted after the removal.

B SPEED PRIOR TRAINING

Architecture. The input sequence is an instruction token followed, for each node, by an image token, a node token and, except after the last node, a query token. The relative pose of the nodes makes windows comparable across domains and setups (Chi et al., 2024). 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 rounded in the DCT domain, where the DCT-II coefficients of each feature along the window are rounded to multiples of 1/50 (Pertsch et al., 2025). Near the ends of an episode, nodes outside the episode are filled with the nearest valid node before rounding. The instruction token is a linear projection of the frozen 1024-dimensional Qwen3-Embedding-0.6B embedding of the instruction at its last token, normalized to unit length (Zhang et al., 2025). Each image token is a linear projection, after RMSNorm, of the frozen DINOv3 ViT-B/16 features of the wrist-camera frame nearest the node at its native resolution, the class token concatenated with the mean patch token (Siméoni et al., 2025). A dropped window replaces its 17 image tokens with a learned token. The query tokens are learned embeddings. The transformer has 12 layers of width 384 with six-head bidirectional attention, query-key RMSNorm, rotary position embeddings over consecutive positions and SwiGLU feed-forward layers, and a learned embedding of the domain label conditions the network. Two linear heads read each query token, one for the arm and one for the gripper.

Optimization. In the first stage, windows of both domains are pooled and sampled uniformly with replacement, so each domain contributes in proportion to its number of windows. We train with AdamW at a learning rate of 10^{-4} after 1,000 warm-up steps, $\beta = (0.9, 0.95)$, no weight decay, gradient clipping at 1.0 and a batch size of 1,024, and keep an exponential moving average of the weights with decay 0.999. For each domain, 5% of the episodes are held out for validation, and we use the checkpoint, among those saved every 5,000 updates, whose quantiles best match the held-out speeds. In the second stage, we fine-tune this prior on each YUBI task for at most 3,000 updates at a learning rate of 5×10^{-6} and keep the best checkpoint.

Interval durations and clock details. We predict speed by classification, which is more robust to noisy targets than squared-error regression (Farebrother et al., 2024). 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 two-hot target splits unit mass between the two bin centers nearest to y_j , and targets beyond the outermost centers are clipped to them. Quantiles are decoded from the cumulative distribution, interpolating linearly within a bin. With the two speeds independent, the cumulative distribution of the duration T_j is the product of those of L_j^a/v_j^a and L_j^g/v_j^g . An actuator that does not move in an interval takes no time. We retimed DROID with the FastUMI-100K label at the guidance coefficient $\lambda = 1$ and the YUBI demonstrations at $\lambda = 2$. The arm travel L_j^a sums Δs_k without its gripper term. The gripper travel L_j^g counts 42.5 mm per unit of opening in every domain, half the stroke of the Robotiq 2F-85, whereas the arc scale ρ_g only sets how far apart the nodes lie. 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

Common settings. Every policy of Section 5 is a $\pi_{0.5}$ policy (Physical Intelligence, 2025) that predicts chunks of $H = 15$ joint-position actions at 15 Hz. Every policy uses train-time RTC (Black et al., 2025b) and is conditioned on a prefix of the first actions of the chunk. Our policy and Filtered BC learn chunks that are compressed in time, from retimed or faster demonstrations, so the same inference delay spans more of their actions, and their prefix covers 0 to 5 actions against 0 to 3 for the default policy. Observation frames are selected on the recorded timing, so the frames that a policy sees do not depend on its 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. Tempo channels are capped at log 5, and a step without motion repeats the channel of the last moving step, or zero before it.

First stage. The default policy, Filtered BC and our policy of Section 5.2 start from the released $\pi_{0.5}$ checkpoint for DROID, whose vision-language backbone stays frozen while the action expert is fine-tuned. Each is trained for up to 20,000 updates with a batch size of 256 and a constant learning rate of 5×10^{-5} after 1,000 warm-up steps, and we evaluate one of the checkpoints saved every 5,000 updates. All three are trained on the measured joint positions of DROID rather than the commanded ones, since the commands are tied to the recorded timing.

Second stage. The YUBI policies of Section 5.3, the default policy, Filtered BC and ours, all start from our policy of the first stage, so that they differ only in their demonstrations and timing, and all their parameters are fine-tuned. Each is trained for 20,000 updates with a batch size of 32 and a learning rate that decays from 5×10^{-5} to 5×10^{-6} on a cosine schedule after 1,000 warm-up steps, and we use its last checkpoint. The joint positions are those that inverse kinematics gives for the tracked gripper poses (Section 5.3).

D EXECUTION IN ROBO LAB

Simulated robot. RoboLab’s DROID scene carries a Franka Panda, which we replace with the Franka Research 3 of our real robot, whose kinematics are the same. The arm takes the link masses, centers of mass and inertias and the joint position limits of the FR3 model in `franka.description`, the firmware velocity limits of Appendix F, which the simulator enforces, and rotor armatures of 0.195 kg m^2 on the first four joints and 0.074 kg m^2 on the last three. The gripper is Robotiq’s PhysX model of the 2F-85, with the finger drive of Robotiq’s simulation guide, the closing speed that DROID commands and a pad friction of 0.7.

Playback at a pace. The policy predicts $H = 15$ joint waypoints on its base clock, one per control period $\delta = 1/15 \text{ s}$. On the tempo channel of quantile q , step h instead lasts $\delta \exp(-\tau_h^{(q)})$, and under a uniform speed-up α , every step lasts δ/α . Starting from the running reference at the step where the chunk takes over, the waypoints form a timed polyline, which the joint reference follows at the 120 Hz physics rate, with each segment’s slope as its velocity and zero acceleration. The gripper takes the command of the step in progress.

Tracking. A computed-torque controller tracks this reference,

$$\mathbf{u} = \mathbf{M}(\mathbf{x})(k_p(\mathbf{x}_{\text{ref}} - \mathbf{x}) + k_d(\dot{\mathbf{x}}_{\text{ref}} - \dot{\mathbf{x}}) + \ddot{\mathbf{x}}_{\text{ref}}) + \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}},$$

with $k_p = 300$ and the critically damped $k_d = 2\sqrt{k_p}$, 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 gripper. Unlike the controller of Section 4, this execution neither plans within the robot’s limits nor joins consecutive chunks with a transition, so the velocity jumps at every waypoint.

Replanning. Every policy runs with RTC, conditioned on the unexecuted actions of the running chunk (Appendix C). The policy is queried when the running chunk reaches the waypoint of its open-loop horizon, 8 steps for the default policy and 6 for ours and Filtered BC, and sees the joint state of that step with the last rendered images. Its chunk takes over two steps later, so that its first two actions are the prefix it was conditioned on. The simulation counts this delay in steps rather than in time, 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 363,567 DROID windows from 3,627 episodes and 273,810 FastUMI-100K windows from 1,827 episodes, whose nodes all lie within the episode, and each window contributes the speed of its center interval. Table 3 compares the predicted distributions with the recorded speeds. The predicted median explains 95% of the variance of held-out log speed on DROID and 93% on FastUMI-100K, and its pinball loss at the median is a sixth of that of an input-blind predictor on DROID and under a quarter on FastUMI-100K. The prior thus reads most of the pace from the path, the images and the instruction. Its quantiles are close to their nominal levels in both domains, with 89.9% and 98.7% of held-out DROID speeds at or below the 0.9 and 0.99 quantiles, and 90.9% and 99.1% on FastUMI-100K.

Tracking jitter. Handheld poses come from visual tracking, whose jitter differs from that of robot joint encoders. In both domains, the energy that the DCT rounding removes from a window falls with speed (Spearman correlation -0.47 on DROID and -0.40 on FastUMI-100K), because slow motion leaves relatively more jitter within a window. A prior that sees this residue could read the absence of tracking jitter in robot data as a cue for the fast handheld pace. The same prior trained without the rounding indeed transfers more speed, raising the median speed of DROID by $1.56\times$ rather than $1.28\times$ when read with the FastUMI-100K label, and its retimed DROID episodes are less smooth, with a SPARC of -3.45 rather than -3.09 under the median clock. It is also less robust to tracking noise: Gaussian jitter of 0.9 mm and 0.35° per axis added to the DROID windows lowers its predicted median speed by 63% and raises its speed-up to $2.04\times$, against 38% and $1.48\times$ with the rounding. With the rounding, the residual of the prior is uncorrelated with the removed energy (Spearman correlation 0.008 on DROID and 0.001 on FastUMI-100K). The rounding thus makes the prior more robust to tracking noise, and the transferred pace mainly reflects how handheld demonstrators move rather than how they are tracked.

Retimed DROID demonstrations. Reading DROID with the FastUMI-100K label raises the predicted speed by $1.28\times$ at the median, $1.38\times$ at the 0.9 quantile and $1.54\times$ at the 0.99 quantile on average. On 2,000 held-out DROID episodes, the median clock shortens the episodes by a factor of 1.54 and the clock of the 0.99 quantile by a factor of 2.66. Their SPARC is -3.09 and -3.07 , against -3.91 for the recorded motion and -3.87 and -3.74 under uniform acceleration to the same durations.

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, $\bar{\mathbf{u}}$ and $\bar{\dot{\mathbf{u}}}$ are the torque and torque rate limits, and the acceleration scale $\tilde{\mathbf{a}} = (\bar{\mathbf{u}} - |\mathbf{g}(\mathbf{x}_b)|) \oslash \text{diag } \mathbf{M}(\mathbf{x}_b)$ is the acceleration that the torque limit leaves after gravity. The planning margins $\beta_v, \beta_a, \beta_u \leq 1$ scale the velocity bound, the acceleration scale and the torque limits.

Table 3: Accuracy of the speed prior on held-out validation windows, each domain read with its own label. 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 18% of the total on DROID and 12% on FastUMI-100K, and the R^2 over all windows, 0.95 and 0.93, is the average of the two columns weighted by these shares.

Domain	Coverage (%)				Pinball, $q = 0.5$		Pinball, $q = 0.9$		R^2 of the median	
	$q = 0.1$	0.5	0.9	0.99	prior	input-blind	prior	input-blind	within	between episodes
DROID	10.3	50.0	89.9	98.7	0.050	0.294	0.021	0.098	0.94	0.97
FastUMI-100K	9.0	49.6	90.9	99.1	0.045	0.204	0.020	0.077	0.92	0.98

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 mm/s and 228 mm/s on DROID and 274 mm/s and 273 mm/s on FastUMI-100K. In the same interval the gripper moves and keeps its source pace. The median clock reads DROID with the FastUMI-100K label, and its speed-up is the geometric mean ratio of its median to the median read with the DROID label.

Distance to gripper event	DROID					FastUMI-100K		
	Share (%)	Recorded	Prior	Median clock	Speed-up	Share (%)	Recorded	Prior
Same interval	16.6	0.24	0.24	0.23	1.08	14.9	0.58	0.58
2 cm to 4 cm	9.1	0.45	0.44	0.57	1.57	10.5	0.67	0.67
6 cm to 10 cm	12.1	0.68	0.68	0.76	1.39	13.4	0.84	0.84
12 cm to 16 cm	8.9	0.84	0.83	0.88	1.32	10.2	0.98	0.98
Transit, none within 16 cm	53.3	1.00	1.00	1.00	1.28	50.9	1.00	1.00

Timing. A worker runs inference and the solver, a 100 Hz client schedules observations and installs plans, and a 1 kHz torque executor plays the active plan, sampled every 1 ms with its progress p , and swaps in the new plan exactly at t_b . Each chunk runs open loop for the horizon of Table 5, 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 $\{0, \dots, 5\}$ for our policy and Filtered BC, whose chunks are compressed in time so that the same latency spans more of their steps, and $\{0, \dots, 3\}$ for the default policy (Appendix C). 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 $T_h^{\min} = \max(\delta/r_h, |c_h - c_{h-1}|/\mu)$, where μ is the gripper speed, so that the arm does not outrun the gripper, and step h_0 at least $T_{h_0}^{\min} = \max((1 - \phi)\delta/r_{h_0}, |c_{h_0} - c_b|/\mu)$, with c_b the gripper command at t_b . The due time t^* 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, $t^* = t_b + T_{h_0}^{\min}$.

DRAFT. The path γ is piecewise quadratic in σ , with the acceleration $\mathbf{a}_m$ constant over rows of 32 ms. It starts at $\mathbf{w}_{h_0}$ with the secant velocity toward $\mathbf{w}_{h_0+1}$ and minimizes $\sum_m \|(\mathbf{a}_m - \mathbf{a}_{m-1}) \oslash \hat{\mathbf{a}}\|^2$ subject to $\|\gamma(h\delta) - \mathbf{w}_h\|_\infty \leq \eta = 5$ 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 γ with knots every 24 ms of σ and retime it with TOPP-RA (Pham & Pham, 2018) on 64

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

- 1: **while** the task runs **do** $\triangleright$ the robot tracks the running plan P
- 2: $o \leftarrow \text{OBSERVE}(P)$ $\triangleright$ at t_o ; with RTC, o includes the unexecuted actions of P
- 3: $(\mathbf{w}_h, c_h, \tau_h^{(q)})_{h=1}^H \leftarrow \pi(o)$, $r_h \leftarrow \exp(\tau_h^{(q)})$ or $r_h \leftarrow \alpha$ $\triangleright t_{\text{vla}}$
- 4: $(t_b, \mathbf{s}_b, h_0, t^*) \leftarrow \text{RESERVE}(P)$ $\triangleright h_0 = \lfloor p(t_b) \rfloor + 1, p(t^*) = h_0$
- 5: $\xi \leftarrow \text{DRAFT}(\mathbf{w}_{h_0:H}, c_{h_0:H}, r_{h_0:H})$ $\triangleright t_{\text{solve}}$ until **INSTALL**
- 6: $\bar{\mathbf{x}} \leftarrow \text{SPLICE}(\mathbf{s}_b, t^*, \xi)$
- 7: $P' \leftarrow \text{SMOOTH}(\mathbf{s}_b, \bar{\mathbf{x}})$
- 8: **if** a step fails or not **VERIFY**($P', \mathcal{L}$) **then**
- 9: $P' \leftarrow \text{BRAKE}(\mathbf{s}_b)$
- 10: **INSTALL**(P', t_b) $\triangleright t_{\text{install}}$, then $P \leftarrow P'$ at t_b
- 11: **function** **DRAFT**($\mathbf{w}, c, r$)
- 12: $\gamma \leftarrow \text{MINJERKPATH}(\mathbf{w}, \eta)$ $\triangleright$ geometry on the base clock σ
- 13: **return** **TIMEOPTIMAL**($\gamma, \mathcal{L}, T^{\min}(c, r)$) $\triangleright$ timing, each step no shorter than $T_h^{\min}$
- 14: **function** **SPLICE**($\mathbf{s}_b, t^*, \xi$)
- 15: start ξ at t^* $\triangleright \xi(t)$ is due at t
- 16: **for** each entry time $t_e \geq t^*$ **do** $\triangleright \xi(t^*) = \mathbf{w}_{h_0}$
- 17: $\psi_e \leftarrow \text{RUCKIG}(\mathbf{s}_b, \xi(t_e), \mathcal{L})$ $\triangleright$ fastest transition, duration D_e
- 18: $\ell_e \leftarrow \max(0, t_b + D_e - t_e)$ $\triangleright$ lateness
- 19: $e^* \leftarrow \arg \text{lexmin}_e(\ell_e, b_e, e)$ $\triangleright b_e$: excess over ϵ at the skipped waypoints
- 20: **return** $\psi_{e^*} \oplus \xi|_{t \geq t_{e^*}}$ $\triangleright \oplus$ concatenates in time; ψ to $\mathbf{w}_{h_0}$ ends no earlier than t^*
- 21: **function** **SMOOTH**($\mathbf{s}_b, \bar{\mathbf{x}}$)
- 22: $P' \leftarrow \text{JERKQP}(\mathbf{s}_b, \bar{\mathbf{x}}, \eta, \mathcal{L})$ $\triangleright$ boxes at the due times t_h on $\bar{\mathbf{x}}$
- 23: **if** P' is infeasible or not **VERIFY**($P', \mathcal{L}$) **then**
- 24: $P' \leftarrow$ the same program with the boxes as a penalty
- 25: **return** P'

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 ms apart from t_b up to step 8, with $\xi(t_e) = \mathbf{w}_{h_0}$ for $t_e \leq t^*$. A transition to $\mathbf{w}_{h_0}$ is stretched so that it arrives no earlier than t^* , unless it would then turn back, and a postponement of t^* for the gripper slows the first span of ξ instead. Ruckig (Berscheid & Kröger, 2021) bounds each transition by $\beta_v \bar{\mathbf{v}}$, by $\beta_a \tilde{\mathbf{a}}$ and by the jerk $\beta_u \tilde{\mathbf{u}} \odot \text{diag } \mathbf{M}(\mathbf{x}_b)$ that the torque rate limit allows. The selected entry is

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

where $\mathcal{S}_e$ holds the waypoints that the entry skips and $\epsilon = 0.05 \text{ rad}$. The rest of ξ 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 ζ_i constant over rows of $T_r = 16 \text{ ms}$, starts from $\mathbf{s}_b$ and minimizes $\sum_i \|T_r \zeta_i \odot \tilde{\mathbf{a}}\|^2$ 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 linearized along $\bar{\mathbf{x}}$, and the torque rate limit $|\mathbf{M}(\bar{\mathbf{x}}_i) \zeta_i| \leq \beta_u \tilde{\mathbf{u}}$. Each waypoint after the entry must lie within η at t_h , and the terminal velocity within 0.05 rad/s of that of ξ . 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 ms and the torque from the full inverse dynamics and its rate every

Table 5: Settings that differ between the setups. The DROID zero-shot tasks of Appendix H are placing bananas and placing cubes.

Setting	YUBI	DROID zero-shot
Gripper	Yubi DYNAMIXEL	Robotiq
Open-loop horizon	8 steps	6 for ours, 8 others
Executed tempo quantiles	0.9, 0.95 and 0.99	0.99

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	$H = 15$ actions, $\delta = 1/15$ s	
Stiffness and damping gains, feedforward κ	$4\times, 2\times, 1$	$1\times, 1\times, 0$
Step cap	none	0.2 rad

8 ms, and is accepted if it exceeds no limit by more than 5% and no joint range by more than 1 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 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 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 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 rad/s^2 .

Low-level execution. The executor applies at 1 kHz the joint torque

$$\mathbf{u} = (\mathbf{J}^\top \mathbf{K}_p^c \mathbf{J} + \mathbf{K}_p)(\mathbf{x}_{\text{ref}} - \mathbf{x}) + (\mathbf{J}^\top \mathbf{K}_d^c \mathbf{J} + \mathbf{K}_d)(\dot{\mathbf{x}}_{\text{ref}} - \dot{\mathbf{x}}) + \mathbf{C}(\mathbf{x}, \dot{\mathbf{x}}) \dot{\mathbf{x}} + \kappa \mathbf{M}(\mathbf{x}) \ddot{\mathbf{x}}_{\text{ref}},$$

clamped to the torque limits, where $(\mathbf{x}_{\text{ref}}, \dot{\mathbf{x}}_{\text{ref}}, \ddot{\mathbf{x}}_{\text{ref}})$ is the current plan sample, $\mathbf{J}$ is the end-effector Jacobian, $\mathbf{K}_p$, $\mathbf{K}_d$ and $\mathbf{K}_p^c$, $\mathbf{K}_d^c$ 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 $\mathbf{K}_p = \text{diag}(40, 30, 50, 25, 35, 25, 10)$, $\mathbf{K}_d = \text{diag}(4, 6, 5, 5, 3, 2, 1)$, $\mathbf{K}_p^c = \text{diag}(400, 400, 400, 15, 15, 15)$ and $\mathbf{K}_d^c = \text{diag}(37, 37, 37, 2, 2, 2)$ 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 Hz.

PD baseline. Following the openpi DROID example (Khazatsky et al., 2024; Physical Intelligence, 2025), the PD baseline holds each waypoint as a position target for $T_h^{\min}$, a 15 Hz stepped reference at $\alpha = 1$, and caps each new target at 0.2 rad from the measured position, as DROID’s joint velocity interface does. It uses the torque law above with the default gains, $\dot{\mathbf{x}}_{\text{ref}} = \ddot{\mathbf{x}}_{\text{ref}} = \mathbf{0}$ 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) rad/s, torque limits (87, 87, 87, 87, 12, 12, 12) Nm and a torque rate limit of 1000 Nm/s. On YUBI, inference and the solve round trip each take 28 ms at the median.

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 s for cups and towel and 60 s for trash. The completion time t_i of a successful episode i is read from the measured gripper. It is the time at which the release that places the second cup 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),

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

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 $t_{\max} = 60$ s otherwise. We report throughput in items per hour (UPH), computed over all N episodes as

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

Smoothness. SPARC measures the smoothness of the end-effector speed (Balasubramanian et al., 2015), where higher values are smoother. We resample the measured flange position of each episode, saved in its step log, at 15 Hz by linear interpolation from its first sample, 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, zero-padded to four times the next power of two of its length,

$$\text{SPARC} = - \sum_{f_k \leq f_*} \sqrt{\left(\frac{\Delta f_k}{f_*} \right)^2 + (\Delta A_k)^2},$$

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 default policy and our policy of Section 5.2, which are fine-tuned 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, on two tasks. The robot has 30 s to put two bananas into a tray and 60 s to put nine gray cubes into a white bin (Figure 3). On both tasks, the default policy runs under the PD baseline and under our controller at $\alpha = 1$, and our policy runs under our controller at $\alpha = 1$ and on its 0.99 tempo channel. Table 5 lists the other settings.

Metrics. We run 30 episodes per condition. The operator marks each item with a key press as it lands in the tray or bin. The first cube episode of the default policy under the PD baseline ran away and was aborted, so that condition keeps the 30 episodes that follow it. An episode succeeds if it places all n items within $t_{\max}$, with $n = 2$ and $t_{\max} = 30$ s for bananas and $n = 9$ and $t_{\max} = 60$ s for cubes. UPH pools the $N = 30$ episodes of a condition,

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

where K_i is the number of items that episode i places within $t_{\max}$, and T_i is the time at which it places the last one if $K_i = n$ and $t_{\max}$ otherwise. SPARC is computed as in Appendix G, from the start of the policy to the last item, or to $t_{\max}$ if the episode does not place all items. SPARC uses the step logs of all 30 episodes, including the cube episode of our policy on the 0.99 tempo channel whose separate controller log was overwritten after a pause.

Results. Table 7 and Figure 5 report the results on bananas. On its 0.99 tempo channel, our policy places both bananas in all 30 episodes, at a UPH of 492.9. Under our controller, the default policy places both in 22 episodes ($p = 0.003$, two-sided Mann-Whitney U test on the number of bananas), at a UPH of 231.7, and under the PD baseline in 17, at 200.7. On its base clock, our policy places both in 29 episodes at a UPH of 376.7. It also moves most smoothly there, with a SPARC of -4.13 against -4.99 for the default policy under our controller ($p < 0.001$), and on the 0.99 tempo channel its SPARC of -4.53 remains above that of the default policy ($p = 0.025$).

Table 8 and Figure 6 report the results on cubes. On its 0.99 tempo channel, our policy places 8.27 cubes on average and all nine in 12 of 30 episodes. Under our controller, the default policy places 5.90 and never all nine ($p < 0.001$), so UPH rises from 354.0 to 511.1. On its base clock, our policy places 7.43 cubes at a UPH of 446.7, and the 0.99 tempo channel raises this to 8.27 ($p = 0.001$). Under the PD baseline, the default policy places only 3.90 cubes, at a UPH of 234.0. Our policy moves most smoothly on its base clock, with a SPARC of -5.46 against -6.26 for the default policy under our controller ($p < 0.001$). On the 0.99 tempo channel, its SPARC of -6.18 is close to that of the default policy ($p = 0.63$).

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 (q99) on the tempo channel of the 0.99 quantile. Bananas is the mean number of bananas placed within 30 s, and Both the number of trials that place both.

Model	Controller	Placing bananas			
		Bananas $\uparrow$	Both $\uparrow$	UPH $\uparrow$	SPARC $\uparrow$
Default	PD	1.57	17	200.7	-5.28
Default	Ours (1x)	1.67	22	231.7	-4.99
Ours	Ours (1x)	1.97	29	376.7	-4.13
Ours	Ours (q99)	2.00	30	492.9	-4.53

Table 8: Real-robot results on the placing cubes task (“put all the gray cubes into the white bin”), with 30 trials per setting. PD is the baseline controller of Appendix F, Ours (1x) runs our controller at $\alpha = 1$ and Ours (q99) on the tempo channel of the 0.99 quantile. Cubes is the mean number of cubes placed within 60 s, and All 9 the number of trials that place all nine.

Model	Controller	Placing cubes			
		Cubes $\uparrow$	All 9 $\uparrow$	UPH $\uparrow$	SPARC $\uparrow$
Default	PD	3.90	0	234.0	-8.19
Default	Ours (1x)	5.90	0	354.0	-6.26
Ours	Ours (1x)	7.43	2	446.7	-5.46
Ours	Ours (q99)	8.27	12	511.1	-6.18

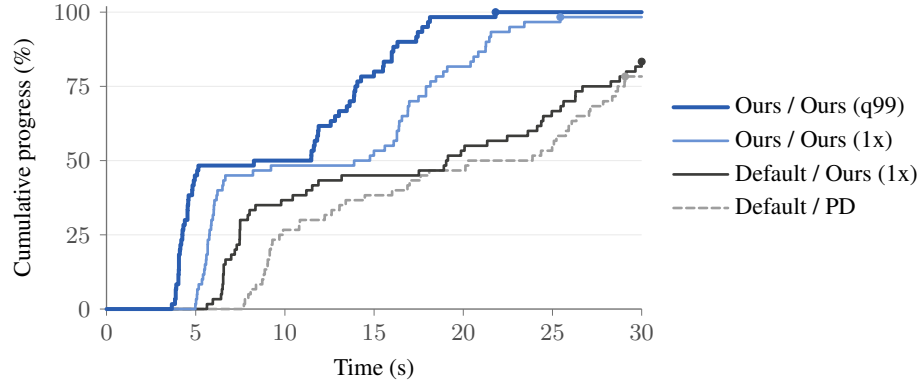


Figure 5: Mean task progress over time on the real-robot placing bananas task, over 30 trials per condition. Each banana placed in the tray within the 30 s limit adds half of a trial, and curves average over trials, so curves further up and to the left progress faster.

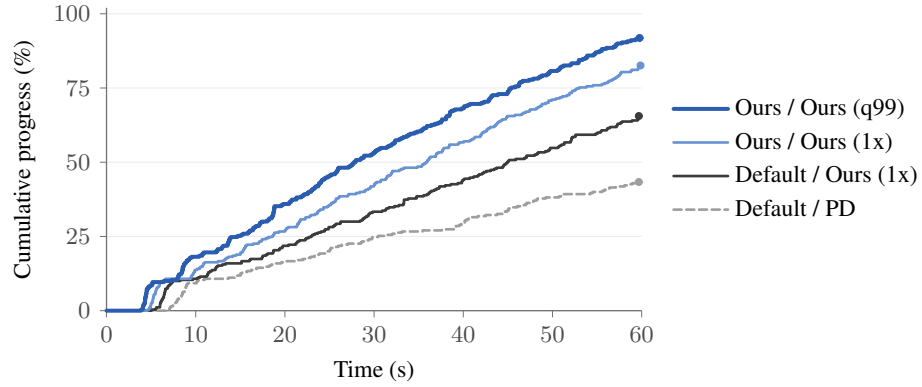


Figure 6: Mean task progress over time on the real-robot placing cubes task, over 30 trials per condition. Each cube placed in the bin within the 60 s limit adds one ninth of a trial, and curves average over trials, so curves further up and to the left progress faster.