$\pi\mathrm{R}^2$: Reactive Real-time Flow Policies

Sungjae Park, Shubham Tulsiani
Carnegie Mellon University
{sungjae2, stulsian}@andrew.cmu.edu

Abstract: Generalist manipulation policies increasingly take the form of action-chunking flow policies built on large pretrained backbones. Such chunks run open-loop, so the policy cannot react to sensory input arriving mid-execution, sacrificing *reactivity*. Replanning more often would restore it, but the perception-to-action pipeline (a large backbone plus multiple denoising steps) is too slow: this *latency* forbids frequent replanning and leaves committed actions stale, making such policies ill-suited for dynamic, closed-loop control. We present $\pi\mathbf{R}^2$, which makes these policies reactive and real-time while retaining large backbones, expressive multi-modal policies, and multi-action prediction. Built on the per-position noise schedule of diffusion forcing, $\pi\mathbf{R}^2$ contributes two ideas. First, it splits conditioning into a fast channel (proprioception, fresh every tick) and an asynchronously updated slow channel (vision-language features), so the policy reacts to proprioception within a chunk while tolerating stale vision. Second, a latency-adaptive flow schedule treats in-flight actions as inpainting conditioning and emits actions in one denoising step per call, letting one trained model adapt to varying hardware latency. Requiring minimal modification to existing architectures, $\pi\mathbf{R}^2$ can be finetuned from a pretrained policy: applied to GR00T-N1.7 on a real xArm6+XHand platform, it replans closed-loop roughly $4\times$ faster than the base policy (25 Hz on an A5000 GPU), acting on a fresh observation every 40 ms. Across simulation and real-world manipulation tasks, $\pi\mathbf{R}^2$ improves the success rate by up to 24% in simulation and 30% in the real world over the strongest baseline.

Project page: https://pi-r2-flow.github.io/

Keywords: Reactivity, Real-time Inference, Flow Policies, Robot Foundation Models

1 Introduction

We have witnessed remarkable progress in training 'robotics foundation models' [1, 2, 3, 4, 5, 6], and three design choices have become ubiquitous across recent efforts. First, **large pre-trained backbones** such as vision-language models (VLMs) [7, 8, 9, 10, 11, 12] are leveraged to process visual (and language) input and extract rich, generalizable representations for downstream action prediction. Second, **expressive policy architectures** such as diffusion [13] and flow matching [14, 15] allow effectively capturing the multi-modal action distributions inherent in diverse human demonstrations. Finally, **action chunking** [16, 17] enables these models to jointly predict multiple actions, providing a richer training signal and improving the temporal consistency of executed actions.

On the one hand, these choices have been crucial in unlocking generalizable and scalable imitation learning. However, they result in *limited reactivity* and *increased latency*, making these foundation models ill-suited for dynamic manipulation tasks that demand fast, smooth, and reactive control. Specifically, the execution of a 'chunk' of predicted actions is performed open-loop, without the ability to react to the sensory input streaming in. While the reactivity can in theory be improved by only executing a small 'sub-chunk' (in the limit just one action) and re-predicting with updated sensory input, naively doing so is infeasible due to the significant computation required in

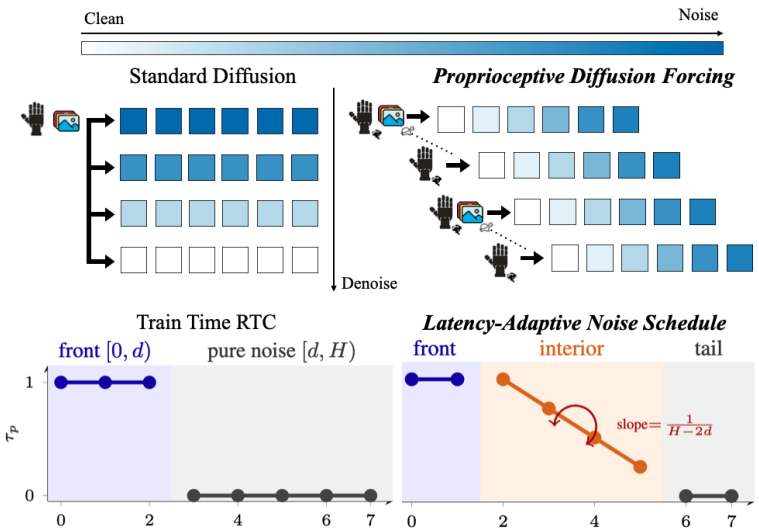


Figure 1: Overview of $\pi\mathbf{R}^2$. **Top.** While standard diffusion/flow matching relies on stale observations to predict actions via iterative denoising, $\pi\mathbf{R}^2$ disentangles the observation into a fast channel (proprioception) and a slow channel (image encoding, VLM embedding, etc.), and uses up-to-date observations for each denoising step. **Bottom.** To incorporate latency and smooth execution, we adopt an adaptive noise schedule, where the action chunk is divided into three regions: clean actions to be taken during inference, actions with increasing noise level following Diffusion Forcing [18], and pure noise with the same length as clean actions. Compared to Train-Time RTC, this enables faster inference and smoother actions.

the 'perception-to-action' prediction – a large visual (and language) processing backbone followed by multiple denoising iterations for flow matching policy inference. This computational bottleneck coupled with action chunking flow matching thus results in two (related) fundamental limitations: a) reduced reactivity, as a large (sub-)chunk of actions must be executed open-loop while the next chunk is predicted, and b) increased latency, as the predicted actions are a function of 'stale' sensory input.

In this work, we develop $\pi\mathbf{R}^2$: a framework that allows reactive real-time policy inference *without* compromising on the design principles of large pre-trained backbones, expressive multi-modal policies, and multi-action prediction. We note that instead of flow (or diffusion) policy inference, which maps uniformly random noise to action chunks over multiple denoising iterations, leveraging the flexible schedule in 'diffusion forcing' [18] offers several benefits: a) the immediate actions to be executed can be at a lower noise level and predicted in fewer iterations, and b) different denoising iterations involved in predicting an action can rely on updated sensory input as available. Although prior work has explored diffusion forcing for policy learning [19], we address the key issues that prevent its widespread adoption for robotics foundation models.

First, while the (immediate) action denoising requires less computation, the backbone (*e.g.,* VLM) feature processing becomes the bottleneck for 'perception-to-action prediction' latency. Our key insight that allows us to circumvent this bottleneck is that not all input modalities need to be processed at the same rate: proprioceptive signals (joint positions, velocities, torques, and contact forces) can be retrieved and processed orders of magnitude faster than images or text. Moreover, for dynamic tasks, proprioception carries sufficient information for *local* reactive corrections (*e.g.*, responding to an unexpected contact or compensating for object slip), while vision and language provide the *global* context needed to guide coarse motion plans. We thus disentangle the conditioning for action denoising into asynchronously updated 'slow' and always up-to-date 'fast' features, effectively allowing high-frequency proprioceptive control within an action chunk. Second, despite our asynchronous diffusion forcing, practical constraints (communication delays, GPU bandwidth) often prevent zero latency. We develop a delay-adaptive noise schedule for diffusion forcing that enables seamless

real-time execution despite latency, allowing the model to continue executing the 'previous' actions while ensuring smooth temporally coherent outputs from subsequent denoising iterations.

Together, these design choices make $\pi\mathbf{R}^2$ roughly $4\times$ faster than the base policy for closed-loop *replanning*, achieving 25 Hz with a large-scale VLA (GR00T N1.7 [3]) on A5000 GPUs. Crucially, this means the policy predicts each action using fresh observation every 40 ms, rather than committing to a long action chunk and replanning only sparsely (*e.g.,* at $\sim$7 Hz). $\pi\mathbf{R}^2$ thus matches the reactivity of simple feedforward policies while retaining the expressivity of flow matching, the training benefits of chunk-based inference, and the semantic grounding of pretrained backbones. We first validate $\pi\mathbf{R}^2$ in simulation where we highlight the benefits of increased reactivity and reduced latency compared to the typical action chunking flow policies. We also show how, instead of training from scratch, we can finetune pre-trained VLAs with a diffusion forcing schedule and demonstrate the efficacy of $\pi\mathbf{R}^2$ for complex real-world manipulation tasks, where we find that it improves over the strongest baseline by up to 24% in simulation and 30% in the real world.

2 Related Work

Action Chunking and Flow Policies. Predicting a *chunk* of future actions rather than a single action, introduced by ACT [16], improves the temporal coherence of imitation-learned policies and has since become standard, including for mobile and contact-rich manipulation [20, 21]. This design has been widely adopted by expressive multi-modal policies based on diffusion [13] and flow matching [22, 23]. For dynamic tasks, however, committing a (sub-)chunk open-loop is suboptimal, as it cannot react to sensory input arriving mid-execution. While our work also reasons over a chunk of actions, its use of diffusion forcing predicts the immediate actions with lower latency and restores reactivity by conditioning successive denoising iterations on progressively fresher sensory input.

Generalist Manipulation Policies. Foundation robotics models also couple chunked flow-matching action heads with large vision-language backbones, yielding vision-language-action (VLA) models that scale imitation learning across diverse tasks and embodiments [24, 25, 5, 26]. Notable examples include π_0/$\pi_{0.5}$ and the dual-system GR00T N1, whose vision-language module feeds a flow-matching action head conditioned on robot proprioception [1, 2, 27]. While this has yielded impressive, generalizable policies, coupling a large backbone with multi-step flow-matching denoising only exacerbates the reactivity problem: the combined per-call latency limits how often the policy can replan. Crucially, this backbone latency would persist even under a diffusion-forcing schedule, since each emitted action still requires a full forward pass through the slow semantic backbone. We instead exploit an asymmetry these models already expose—proprioception is far cheaper to process than vision and language—refreshing proprioceptive conditioning at every control tick while letting the slow vision-language features update asynchronously, making this same class of models reactive.

Real-time Execution of Action-Chunking Policies. The high inference latency of modern VLAs has motivated methods for executing action chunks smoothly while the next prediction is computed [28]. RTC maintains continuity across asynchronous chunks via inference-time inpainting on a frozen action prefix, while Training-Time RTC instead folds this prefix conditioning into training, avoiding the extra inference-time cost [29, 30]. Streaming Diffusion Policy and Streaming Flow Policy emit actions incrementally from a diffusion-forcing-style variable-noise buffer maintained across observations [19, 31]. None, however, address the latency of repeatedly conditioning on a large semantic backbone; Streaming Diffusion Policy, for instance, uses compact visuomotor policies and runs synchronously. Concurrent to our work, FASTER [32] similarly reduces reaction latency in flow-based VLAs via a diffusion-forcing-inspired horizon-aware schedule with action-prefix conditioning, but forwards the backbone once per chunk and conditions all actions on a single fixed observation, improving latency and smoothness without improving reactivity to sensory input arriving during execution. $\pi\mathbf{R}^2$ instead refreshes proprioceptive conditioning within the chunk, reacting to incoming feedback as actions execute while vision-language features update asynchronously.

3 $\pi\mathbf{R}^2$

Our goal is to modify existing large-scale flow policies for higher reactivity and lower inference latency. We first review the building blocks – flow matching and diffusion forcing – and then introduce

two modifications that together yield a real-time, closed-loop flow policy. The modifications require only a flow matching action head conditioned on a heavy pretrained backbone, so they apply equally to world-action models [33, 34, 35, 36]; we instantiate and evaluate on VLAs, the most common member of this family.

3.1 Preliminary

3.1.1 Flow Matching and Action Chunking Policy

We build on flow matching [37, 38] with action chunking [16, 17]. Flow matching learns a velocity field $v_\theta(\mathbf{x}_t, t)$ that transports samples from noise $p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I})$ to data $p_1 = p_{\text{data}}$ along the conditional interpolation $\mathbf{x}_t = (1-t)\,\boldsymbol{\epsilon} + t\,\mathbf{x}_1$, $t \in [0,1]$, trained against the conditional velocity $u_t(\mathbf{x}_t \mid \mathbf{x}_1) = \mathbf{x}_1 - \boldsymbol{\epsilon}$:

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t,\mathbf{x}_1,\boldsymbol{\epsilon}}\left[\|v_\theta(\mathbf{x}_t, t) - (\mathbf{x}_1 - \boldsymbol{\epsilon})\|^2\right]. \tag{1}$$

In the action-chunking policy setting, $\mathbf{x}_1 = (\mathbf{a}_t, \ldots, \mathbf{a}_{t+H})$ is a chunk of H future actions conditioned on observation $\mathbf{o}_t$, generated jointly by K denoising steps; the robot executes the first $h \leq H$ actions before re-planning. **Crucially, all H positions share a single noise level t and the chunk is committed open-loop: the h executed actions never account for new observations during their execution window**, limiting the reactivity that manipulation tasks demand.

3.1.2 Diffusion Forcing and Streaming Diffusion

Diffusion forcing [18, 39] generalizes flow matching by assigning each chunk position $p \in \{0, \ldots, H-1\}$ an independent noise level $\tau_p \in [0,1]$:

$$\mathbf{x}_{\tau,p} = (1-\tau_p)\,\boldsymbol{\epsilon}_p + \tau_p\,\mathbf{a}_p, \tag{2}$$

and the model $v_\theta(\mathbf{x}_\tau, \boldsymbol{\tau}, \mathbf{o})$ predicts a per-position velocity. The flexibility of position-dependent noise levels enables *closed-loop* action prediction by conditioning successive denoising steps on progressively more recent observations within a single chunk. *Streaming diffusion* [19] is one such instantiation: it imposes a linearly increasing noise schedule across the chunk so that the leading positions are fully denoised within a few steps, and each denoising step incorporates a fresh observation.

In this work, we also adopt diffusion forcing, focusing on an underexplored regime: enabling large-scale flow policies (*e.g.,* VLAs) to be more reactive and closed-loop, where large latency naturally arises from its model size and components. The following two subsections present the architectural and scheduling modifications that make diffusion forcing a practical real-time, reactive policy on top of any flow matching action head.

3.2 Proprioception-Reactive Diffusion Forcing

A VLA processes the observation $\mathbf{o}_t = (\mathbf{s}_t, \mathbf{I}_t, \mathbf{T}_t)$ – proprioception $\mathbf{s}_t$ (joint positions, end-effector pose, *etc.*), image $\mathbf{I}_t$, and language $\mathbf{T}_t$ – to predict an action chunk $(\mathbf{a}_t, \ldots, \mathbf{a}_{t+H})$. Internally, the inputs are preprocessed (dominated by image processing) and passed through the large backbone(*e.g.,* VLM), producing language-aligned features; proprioception is processed by a small MLP into a state embedding; the DiT action head then conditions on the concatenated representation and runs K denoising steps over the chunk. In practice, these components differ greatly in cost. On RTX A6000 with GR00T-N1.7, image preprocessing + VLM (~ 60 ms) plus DiT denoising ($K{=}4$ steps, ~ 80 ms) sum to ~ 140 ms per call – ~ 7 control ticks at 50 Hz. This motivates disentangling the slow VLM path from the fast action prediction loop.

We propose to disentangle the DiT's conditioning (Fig. 1, left) into a *slow channel* (vision and language features from the VLM and image/text encoders) and a *fast channel* (proprioception, fresh every tick). This split mirrors the structure of manipulation itself: vision and language provide coarse spatial and task guidance, while fresh proprioception drives the fine motor refinement that precision demands. Diffusion forcing additionally reduces the per-call denoising work: because

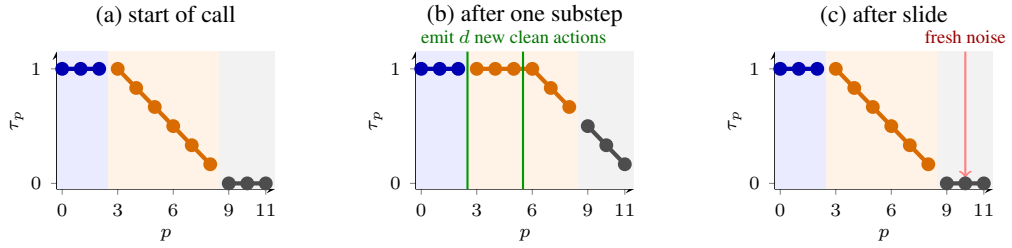


Figure 2: **Inference cycle, one call.** (a) Buffer at $\tau^{\star,d}$ at the start of the call. (b) One Euler substep (one NFE) shifts the schedule right by d slots: positions $[d, 2d)$ reach $\tau{=}1$ and are released, and the ramp extends through the back (c) The buffer slides d positions: just-emitted actions become the new front conditioning, d fresh-noise slots ($\tau{=}0$) are appended – the schedule reproduces exactly.

the front of the chunk is kept near $\tau = 1$, a single denoising step per call suffices to produce clean actions at the front (one or more, depending on the schedule; see Sec. 3.3), versus K steps in standard chunked denoising. Combining the two, the DiT action head runs one denoising step against fresh proprioception and a *cached* slow feature; vision and language are processed asynchronously in a background thread (image preprocessing + VLM forward) and refresh the cache when complete, so the action head conditions on a slow feature of age d_{vlm} ticks (the vision/text wall-clock latency). Per-call cost of the action head is therefore one NFE only, independent of backbone size.

To absorb the resulting slow-channel staleness, we delay the slow channel at training time by $d_{\text{vlm}} \sim \text{Uniform}\{0, \dots, d_{\text{vlm}}^{\max}\}$ and supply the delay value through a learned embedding indexed by the integer delay, added to the slow representation; at deploy time the measured d_{vlm} uses the same embedding. The resulting policy is *proprioception-reactive*: it reacts to fresh joint state every denoising step while tolerating bounded staleness on vision and language, recovering closed-loop control even behind a large VLA backbone. We refer to this architecture variant as *Proprioception-Reactive Diffusion Forcing*.

3.3 Latency-Adaptive Flow Schedule

By combining diffusion forcing with asynchronous vision/text processing and single-step DiT inference, the per-call delay d of the action prediction loop (distinct from the vision-language staleness d_{vlm} of Sec. 3.2) reduces to one denoising step of the action head. However, in reality, d varies: it depends on each component size and inference GPU, includes network latency when policy and robot run on separate machines (e.g., Ethernet round-trips in remote-inference setups), and fluctuates with computational load and per-call jitter. The policy must therefore handle a range of inference delays $d \in [1, d_{\max}]$.

A naive linearly increasing noise schedule, as in standard streaming diffusion [19], does not address this regime: it implicitly assumes $d = 0$ (instantaneous inference, no in-flight actions), so when applied at $d > 0$ the returning chunk can jump at the chunk boundary [29, 40]. We address both issues with a per-position noise schedule parameterized by the inference delay d: a clamped-clean front carries the d in-flight actions as inpaint conditioning, a ramped interior emits d clean actions per call at the cost of one denoising step, and randomizing d at training lets a single model adapt to whatever value is measured at each call.

Staircase schedule (Fig. 1, right). For a target inference delay d, $\pi\mathbf{R}^2$ uses the three-region staircase $\boldsymbol{\tau}^{\star,d} \in [0,1]^H$

$$\tau_p^{\star,d} = \begin{cases} 1 & 0 \le p < d \\ 1 - \dfrac{p-d}{H-2d} & d \le p < H-d \\ 0 & H-d \le p \le H-1 \end{cases} \tag{3}$$

with interior slope $s = 1/(H{-}2d)$. Three roles by position: **front** $[0, d)$, the in-flight actions, clamped clean as inpaint conditioning; **interior** $[d, H{-}d)$, a linear ramp from clean to noise; **tail**

$[H-d, H)$, the d pure-noise slots appended at the back per cycle. This staircase can be viewed as a diffusion-forcing generalization of training-time RTC [30]: both clamp a clean front of d in-flight actions for inpaint conditioning, but RTC applies a single shared noise level over the remaining $H - d$ positions, whereas the staircase replaces it with a ramped interior plus a pure-noise tail to enable single-step emission under diffusion forcing.

Training. At each batch we sample $d \sim \mathrm{Uniform}\{1, \ldots, d_{\max}\}$ and build $\boldsymbol{\tau}^{\star,d}$. The front d slots are filled by the ground-truth actions and excluded from the loss via $m_p = \mathbf{1}[p \geq d]$ (mirroring inference-time inpaint conditioning); the interior and tail are noised at the staircase levels and contribute to the per-position MSE. We additionally apply symmetric jitter $\tau_p \leftarrow \mathrm{clip}(\tau_p + \delta_p, 0, 1)$ with $\delta_p \sim \mathrm{Uniform}[-j, j]$ to absorb small deviations from the central staircase that arise from per-call d variation. With probability $p = 0.2$, we instead train on a standard flow schedule (single $\tau \sim \mathrm{Uniform}[0, 1]$ shared across positions, no mask), so the same network can also denoise a full chunk from pure noise, used at inference to warm-start the buffer at episode start.

Compatibility with existing flow policy action heads. The procedure above plugs into any DiT-style flow matching action with a one-line architectural change: the AdaLN conditioning (which modulates each DiT block by the noise level τ) becomes per-position, with one (γ_p, β_p) pair per chunk position rather than one shared across positions. All other components (attention, MLP, pretrained large backbone, slow/fast feature paths) remain unchanged, so existing pretrained policies, such as VLAs, can be fine-tuned with $\pi\mathbf{R}^2$ without touching the backbone.

Inference cycle (Fig. 2). At episode start we warm up the buffer with standard-flow inference (denoising all H positions from pure noise) and re-noise to match $\boldsymbol{\tau}^{\star,d}$ for the initial measured d. From then on, each call applies one denoising step that updates each position by

$$\mathbf{x}_p \;\leftarrow\; \mathbf{x}_p + \Delta\tau_p \cdot v_\theta(\mathbf{x}, \boldsymbol{\tau}, \mathbf{o})_p, \qquad \tau_p \;\leftarrow\; \tau_p + \Delta\tau_p, \tag{4}$$

with region-dependent per-position advances $\Delta\tau_p$ chosen so the schedule shifts right by d slots: positions $[d, 2d)$ reach $\tau{=}1$ and are emitted, and the rest of the buffer rotates forward (Fig. 2b). The buffer then slides by d positions, with d fresh-noise slots ($\tau{=}0$) appended at the back (Fig. 2c). The schedule reproduces exactly when d holds steady; when d changes between calls the $\Delta\tau_p$ pattern adapts to the new measured d, and the buffer is pulled toward the new $\boldsymbol{\tau}^{\star,d}$ over a few calls.

4 Experiments

4.1 Simulation Experiments

Setup. We first test $\pi\mathbf{R}^2$ on the Leap Cube Reorientation task in MuJoCo Playground [41], a common task that requires reactivity to keep the cube from falling and to achieve the target cube pose. Demonstrations are collected at 50 Hz from 4 RL experts (different seeds), yielding 200 trajectories total. All methods are trained as state-based flow policies, with prediction chunk length $H = 16$. We run two studies: an execution-horizon h sweep under zero inference delay (Sec. 4.1.1), and a deployment study under a fixed latency budget (Sec. 4.1.2).

4.1.1 Execution-horizon (h) sweep without inference delay

We first study the setting where inference delay $d = 0$, to isolate two things: (i) the tradeoff between task performance and execution horizon h (executing more actions open-loop misses the recent state), and (ii) whether $\pi\mathbf{R}^2$'s amortized denoising, where each action's final denoising step conditions on the most recent observation, matches the reactivity of $h{=}1$ flow at a fraction of the per-call cost. At evaluation, standard flow matching uses 16 denoising steps to predict the whole chunk, whereas $\pi\mathbf{R}^2$ uses one denoising step and emits one action at a time. We sweep $h \in \{1, 2, 4, 8\}$ for standard flow; $\pi\mathbf{R}^2$ emits one action per call ($h = 1$, 1 NFE per call).

Results. Results are highlighted in Fig. 3 (left). Intuitively, the performance of standard flow degrades as h grows, confirming the h tradeoff for reactive tasks. Meanwhile, $\pi\mathbf{R}^2$ matches the flow

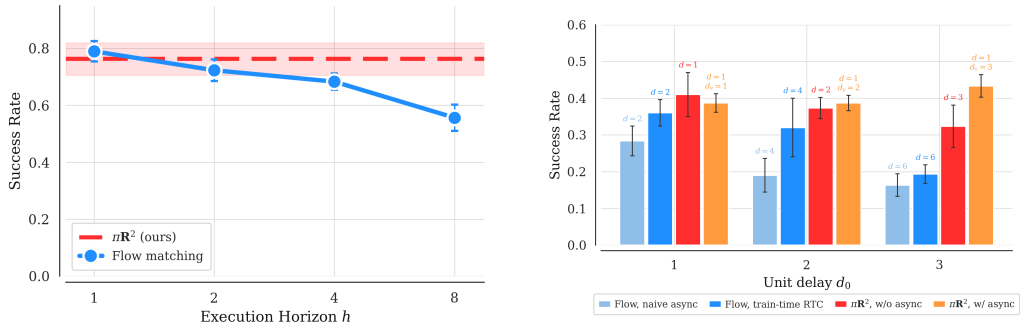


Figure 3: Simulation results. **Left.** Without any inference delay, executing a smaller number of actions within the chunk benefits the performance for a flow matching policy. $\pi\mathbf{R}^2$ also achieves the same performance via replanning every timestep, while the only last denoising step is conditioned on up-to-date state. **Right.** When there is an inference delay for each component, $\pi\mathbf{R}^2$ reduces the effective delay d by reducing the number of denoising steps and asynchronously processing the visual features. For each datapoint, d indicates the effective delay of proprioception, while d_v is visual delay. We train 3 policies with different seeds for each method and report the mean and std of the success rate.

configuration of standard flow $h \in \{1, 2\}$. This supports that leveraging fresh observation more frequently is beneficial for such reactive tasks, and amortizing the denoising budget across calls does not sacrifice quality, even though each call costs only one NFE. We next introduce realistic latency and test each deployment regime under that constraint.

4.1.2 Deployment under VLA-level inference delay

To consider a realistic latency, we take into account GR00T-N1.7 computation cost (Sec. 3.2: vision/text VLM processing ≈ 60 ms, $K{=}4$ NFE denoising ≈ 80 ms) and define unit delay d_0 as the $K = 4$ denoising cost, resulting in vision/text processing $\approx 0.75\, d_0$ and one NFE $= d_0/4$.

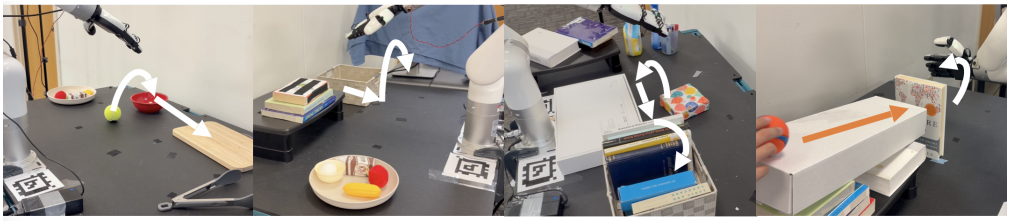
Baselines. We compare four methods: (i) *naive-async* runs the policy inference for predicting the new chunk while executing previous chunk. The inference does not take into account the actions being executed during inference. (ii) *train-time RTC* [30] explicitly conditions the in-flight actions while predicting the new chunk, resulting in smooth chunk boundaries. (iii) $\pi\mathbf{R}^2$ without asynchronous fast/slow channel processing only applies latency-adaptive flow schedule (Sec. 3.3) without asynchronously processing the fast and slow channel, and (iv) $\pi\mathbf{R}^2$ with asynchronous processing applies all proposed modifications.

The effective per-call delays are then $1.75\, d_0$ (end-to-end inference: *naive-async* and *train-time RTC*), $1.0\, d_0$ ($\pi\mathbf{R}^2$ w/o async), and $0.25\, d_0$ ($\pi\mathbf{R}^2$ w async). Setting $d_0 \in \{1, 2, 3\}$ control ticks and ceil-rounding gives effective proprioception delay $d \in \{2, 4, 6\}/\{1, 2, 3\}/\{1, 1, 1\}$, respectively. For $\pi\mathbf{R}^2$ with async processing, we additionally delay the object pose by $d_{vis} \in \{1, 2, 3\}$. For all methods, we set execution horizon $h = d$ across all methods.

Results. Fig. 3 (right) reports the mean over three training seeds (error bars show std). $\pi\mathbf{R}^2$ w/ async wins at every d_0 (0.39, 0.39, 0.43), beating naive-async (0.28, 0.19, 0.16) and Train-time RTC (0.36, 0.32, 0.19) with margins that widen as delay grows. The reason is effective delay: baselines pay $d{=}\lceil 1.75\, d_0 \rceil$ per call while $\pi\mathbf{R}^2$ pays $d{=}d_0$ (w/o async) or $d{=}1$ (w/ async). Async processing lets us hold the action delay at 1 while only the visual delay d_{vis} grows, so performance degrades gracefully with d_{vis} – supporting the view that vision provides coarse guidance while up-to-date proprioception drives manipulation.

4.2 Real World Experiments

We now mirror the deployment study above on a real xArm6 + XHand setup by fine-tuning GR00T-N1.7 model [27] for dexterous manipulation. Demos are collected at 25 Hz (capped by the teleoperation recording pipeline), so we report d in 25-Hz ticks (1 tick $\approx$ 40 ms); We use RTX A5000



***Task 1:** Don't Spill* ***Task 2:** Tidy Up Book* ***Task 3:** Insert Box* ***Task 4:** Catch Book*

Figure 4: **Real-world manipulation tasks.** Arrows in each image show the desired task outcome.

Table 1: **Real World Results.** (xArm6 + XHand, fine-tuned GR00T N1.7). d is the measured wall-clock inference delay in 25-Hz control ticks (1 tick $\approx$ 40 ms), which is the same as execution horizon h for all methods other than synchronous inference. For each task we report success rate (SR) over N trials and a partial-progress score (Prog) capturing the fraction of subgoals completed per episode.

	Don't Spill		Tidy up Book		Insert Box		Catch Book
Setting	SR	Prog	SR	Prog	SR	Prog	SR
Flow, Synchronous ($h = 10$)	4/20	16/80	4/20	9/40	11/20	56/80	4/20
Flow, Naive Async, dense, TE [16]	7/20	30/80	7/20	15/40	12/20	61/80	2/20
Flow, Train-Time RTC [30]	9/20	45/80	8/20	18/40	10/20	53/80	5/20
$\pi\mathbf{R}^2$	**10/20**	**55/80**	**12/20**	**24/40**	**16/20**	**68/80**	**11/20**

for both training and inference, resulting in $d = 4 \sim 5$ for baselines and $d = 1 \sim 2$ for $\pi\mathbf{R}^2$. We include arm and hand joint angles, along with hand per-joint torque and fingertip forces in the proprioceptive state.

Tasks. We evaluate our method on four contact-rich dexterous, reactive manipulation tasks, shown in Fig. 4. *Don't Spill* requires placing a ball into a bowl and moving the bowl onto a cutting board without dropping the ball. *Tidy Up Book* requires extracting a book from a pile, grasping it, and placing it into a basket. *Insert Box* requires pushing a box against a wall to stand it upright before inserting it between a row of books. *Catch Book* requires the robot reacting to the falling book (induced by the ball hitting the book) and grasping it without dropping it.

All four tasks demand highly reactive behavior. Picking up the ball requires dynamic grasping, moving the bowl without tilting it relies on proprioceptive feedback, and both extracting a book from a pile and pushing a box upright require precise contact-rich interactions and reactive control. Lastly, catching a book requires the robot to quickly react so that the book stays within its palm. We collect 200, 300, 300, and 100 teleoperated demonstrations for each task, respectively.

Baselines. We compare $\pi\mathbf{R}^2$ against three baselines, all fine-tuned from GR00T-N1.7 with same training budget. (1) *Flow, Synchronous* stalls the robot for the full $d{=}4 \sim 5$ ticks while inference runs, then executes the predicted chunk ($h = 10$). (2) *Flow, Naive Async (dense, TE)* [21] runs the policy asynchronously without conditioning on in-flight actions: *dense* re-issues a query every d ticks (whenever the previous call returns), and *TE* (temporal ensembling) [16] for overlapping action predictions for the same timestep. (3) *Flow, Train-Time RTC* [30] additionally trains the policy to inpaint a clean d-action front, anchoring the new chunk to the executed history at the chunk boundary.

Results. Table 1 summarizes the experimental results. $\pi\mathbf{R}^2$ with asynchronous vision–language inference operates near the per-control-tick limit ($d{=}1$ at 25 Hz, with occasional increases to $d{=}2$ under network delays), whereas all flow-based baselines incur the full GR00T pipeline latency of $d \in 4, 5$. Synchronous inference produces jittery motion at chunk boundaries because the robot pauses while waiting for the next action chunk, leading to failures such as the ball rolling off the table. Naive asynchronous inference combined with temporal ensembling yields smoother motion but sacrifices execution precision. Train-Time RTC [30] is the strongest baseline; however, $\pi\mathbf{R}^2$ outperforms it

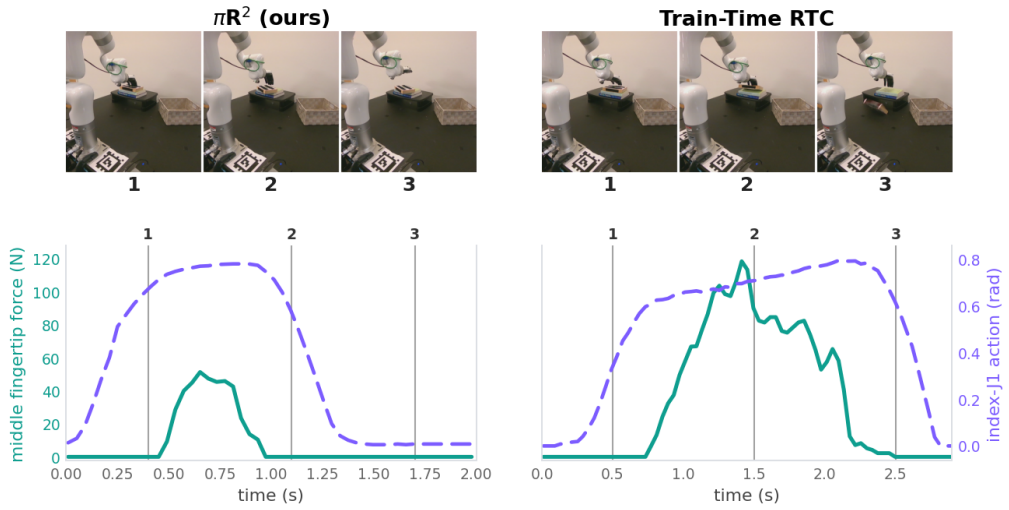


Figure 5: $\pi\mathbf{R}^2$ **reacts to proprioception, while baselines run a stale plan** (*Tidy Up Book*). Fingertip force (solid, left axis) and the emitted action (dashed, right axis) over time, for $\pi\mathbf{R}^2$ and Train-Time RTC; numbered markers link plot times to the overhead frames above. $\pi\mathbf{R}^2$ grips just enough ($\sim$50 N), while RTC reacts late and over-grips to $\sim$120 N, dropping the book.

across all metrics, with the largest gains on reactivity-critical tasks such as Tidy Up Book, Insert Box, and Catch Book, achieving approximately $20 \sim 30\%$ absolute improvement. We observe that RTC also struggles to recover from failures because previously generated in-flight actions bias the policy toward continuing its recent motion, which reduces its ability to react promptly, particularly in Insert Box. Qualitatively, $\pi\mathbf{R}^2$ produces smoother and more accurate motions, even during failure recovery. These results demonstrate the value of the real-time proprioceptive feedback enabled by $\pi\mathbf{R}^2$.

Reactivity analysis. Fig. 5 plots a fingertip contact force against the action each method emits over time on *Tidy Up Book*. Because $\pi\mathbf{R}^2$ refreshes proprioception at every denoising step, it modulates its grip from live force feedback and stops near $\sim$50 N, gripping just enough to reorient the book. Train-Time RTC instead commits to a stale plan and reacts late, so it keeps pushing down after contact, and its middle-finger force overshoots to $\sim$120 N, crushing the book. This force modulation is the mechanism behind our quantitative gains: reacting to contact as it happens rather than after the current chunk finishes. Tmaihe same pattern holds in all remaining tasks (Appendix A.4).

5 Discussion

We presented $\pi\mathbf{R}^2$, a principled modification to existing VLA architectures that yields real-time, closed-loop policies through two orthogonal contributions: asynchronous vision/text processing that decouples the slow VLM backbone from the action loop, and a latency-adaptive flow schedule that absorbs variable inference delay via a per-position noise schedule parameterized by d. Together, they reduce per-call delay by up to $4\times$ while remaining compatible with any VLA action head. This enables $\pi\mathbf{R}^2$ to handle highly reactive behaviors – non-prehensile, contact-rich manipulation – where baselines struggle, both in simulation and on real hardware.

Limitations

Our approach has several limitations. First, we do not address sources of latency external to the model itself, such as communication delays between the inference server and the robot client. Second, we keep the base policy architecture unchanged; designing it to emphasize proprioceptive fea-

tures more strongly (e.g., dedicated attention heads for proprioceptive tokens) could further amplify reactivity, and we leave this direction to future work.

Acknowledgements

We appreciate the helpful discussions with CISCO, Tony Tao, Andrew Wang, Jason Liu, and Yuxuan Kuang. We also thank Yishu Li, Kallol Saha, and Soumojit Bhattacharya for helping real-world experiments. Lastly, we thank Hyeonwoo Kim for the feedback on website design. This work was supported by gift awards from CISCO, Google, and NSF Award IIS-2442282.

References

[1] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. $\pi 0$: A vision-language-action flow model for general robot control. In *RSS*, 2025.

[2] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, M. Y. Galliker, D. Ghosh, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, D. LeBlanc, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, A. Z. Ren, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, J. Tanner, Q. Vuong, H. Walke, A. Walling, H. Wang, L. Yu, and U. Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

[3] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[4] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. In *RSS*, 2023.

[5] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *CoRL*, 2023.

[6] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. In *CoRL*, 2024.

[7] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.

[8] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

[9] S. Bai, Y. Cai, R. Chen, K. Chen, X. Chen, Z. Cheng, L. Deng, W. Ding, C. Gao, C. Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[10] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[11] L. Beyer, A. Steiner, A. S. Pinto, A. Kolesnikov, X. Wang, D. Salz, M. Neumann, I. Alabdulmohsin, M. Tschannen, E. Bugliarello, T. Unterthiner, D. Keysers, S. Koppula, F. Liu, A. Grycner, A. Gritsenko, N. Houlsby, M. Kumar, K. Rong, J. Eisenschlos, R. Kabra, M. Bauer,

M. Bošnjak, X. Chen, M. Minderer, P. Voigtlaender, I. Bica, I. Balazevic, J. Puigcerver, P. Papalampidi, O. Henaff, X. Xiong, R. Soricut, J. Harmsen, and X. Zhai. Paligemma: A versatile 3b vlm for transfer, 2024. URL https://arxiv.org/abs/2407.07726.

[12] M. Deitke, C. Clark, S. Lee, R. Tripathi, Y. Yang, J. S. Park, M. Salehi, N. Muennighoff, K. Lo, L. Soldaini, J. Lu, T. Anderson, E. Bransom, K. Ehsani, H. Ngo, Y. Chen, A. Patel, M. Yatskar, C. Callison-Burch, A. Head, R. Hendrix, F. Bastani, E. VanderBilt, N. Lambert, Y. Chou, A. Chheda, J. Sparks, S. Skjonsberg, M. Schmitz, A. Sarnat, B. Bischoff, P. Walsh, C. Newell, P. Wolters, T. Gupta, K.-H. Zeng, J. Borchardt, D. Groeneveld, C. Nam, S. Lebrecht, C. Wittlif, C. Schoenick, O. Michel, R. Krishna, L. Weihs, N. A. Smith, H. Hajishirzi, R. Girshick, A. Farhadi, and A. Kembhavi. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models, 2024. URL https://arxiv.org/abs/2409.17146.

[13] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *IJRR*, 2023.

[14] C. Yuan, C. Wen, T. Zhang, and Y. Gao. General flow as foundation affordance for scalable robot learning. *arXiv preprint arXiv:2401.11439*, 2024.

[15] C. Pan, G. Anantharaman, N.-C. Huang, C. Jin, D. Pfrommer, C. Yuan, F. Permenter, G. Qu, N. Boffi, G. Shi, et al. Much ado about noising: Dispelling the myths of generative robotic control. *arXiv preprint arXiv:2512.01809*, 2025.

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

[17] T. T. Zhang, D. Pfrommer, C. Pan, N. Matni, and M. Simchowitz. Action chunking and exploratory data collection yield exponential improvements in behavior cloning for continuous control. *arXiv preprint arXiv:2507.09061*, 2025.

[18] B. Chen, D. Martí Monsó, Y. Du, M. Simchowitz, R. Tedrake, and V. Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.

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

[20] Z. Fu, T. Z. Zhao, and C. Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation, 2024. URL https://arxiv.org/abs/2401.02117.

[21] T. Z. Zhao, J. Tompson, D. Driess, P. Florence, K. Ghasemipour, C. Finn, and A. Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024.

[22] Q. Zhang, Z. Liu, H. Fan, G. Liu, B. Zeng, and S. Liu. Flowpolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation, 2024. URL https://arxiv.org/abs/2412.04987.

[23] G. Yan, J. Zhu, Y. Deng, S. Yang, R.-Z. Qiu, X. Cheng, M. Memmel, R. Krishna, A. Goyal, X. Wang, and D. Fox. Maniflow: A general robot manipulation policy via consistency flow training, 2025. URL https://arxiv.org/abs/2509.01819.

[24] O. X.-E. Collaboration, A. O'Neill, A. Rehman, A. Gupta, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, A. Tung, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Gupta, A. Wang, A. Kolobov, A. Singh, A. Garg, A. Kembhavi, A. Xie, A. Brohan, A. Raffin, A. Sharma, A. Yavary, A. Jain, A. Balakrishna, A. Wahid, B. Burgess-Limerick, B. Kim, B. Schölkopf, B. Wulfe, B. Ichter, C. Lu, C. Xu, C. Le, C. Finn, C. Wang, C. Xu, C. Chi, C. Huang, C. Chan, C. Agia, C. Pan, C. Fu, C. Devin, D. Xu, D. Morton, D. Driess, D. Chen, D. Pathak, D. Shah, D. Büchler, D. Jayaraman, D. Kalashnikov, D. Sadigh, E. Johns, E. Foster, F. Liu, F. Ceola, F. Xia, F. Zhao,

F. V. Frujeri, F. Stulp, G. Zhou, G. S. Sukhatme, G. Salhotra, G. Yan, G. Feng, G. Schiavi, G. Berseth, G. Kahn, G. Yang, G. Wang, H. Su, H.-S. Fang, H. Shi, H. Bao, H. B. Amor, H. I. Christensen, H. Furuta, H. Bharadhwaj, H. Walke, H. Fang, H. Ha, I. Mordatch, I. Radosavovic, I. Leal, J. Liang, J. Abou-Chakra, J. Kim, J. Drake, J. Peters, J. Schneider, J. Hsu, J. Vakil, J. Bohg, J. Bingham, J. Wu, J. Gao, J. Hu, J. Wu, J. Wu, J. Sun, J. Luo, J. Gu, J. Tan, J. Oh, J. Wu, J. Lu, J. Yang, J. Malik, J. Silvério, J. Hejna, J. Booher, J. Tompson, J. Yang, J. Salvador, J. J. Lim, J. Han, K. Wang, K. Rao, K. Pertsch, K. Hausman, K. Go, K. Gopalakrishnan, K. Goldberg, K. Byrne, K. Oslund, K. Kawaharazuka, K. Black, K. Lin, K. Zhang, K. Ehsani, K. Lekkala, K. Ellis, K. Rana, K. Srinivasan, K. Fang, K. P. Singh, K.-H. Zeng, K. Hatch, K. Hsu, L. Itti, L. Y. Chen, L. Pinto, L. Fei-Fei, L. Tan, L. J. Fan, L. Ott, L. Lee, L. Weihs, M. Chen, M. Lepert, M. Memmel, M. Tomizuka, M. Itkina, M. G. Castro, M. Spero, M. Du, M. Ahn, M. C. Yip, M. Zhang, M. Ding, M. Heo, M. K. Srirama, M. Sharma, M. J. Kim, M. Z. Irshad, N. Kanazawa, N. Hansen, N. Heess, N. J. Joshi, N. Suenderhauf, N. Liu, N. D. Palo, N. M. M. Shafiullah, O. Mees, O. Kroemer, O. Bastani, P. R. Sanketi, P. T. Miller, P. Yin, P. Wohlhart, P. Xu, P. D. Fagan, P. Mitrano, P. Sermanet, P. Abbeel, P. Sundaresan, Q. Chen, Q. Vuong, R. Rafailov, R. Tian, R. Doshi, R. Mart'in-Mart'in, R. Baijal, R. Scalise, R. Hendrix, R. Lin, R. Qian, R. Zhang, R. Mendonca, R. Shah, R. Hoque, R. Julian, S. Bustamante, S. Kirmani, S. Levine, S. Lin, S. Moore, S. Bahl, S. Dass, S. Sonawani, S. Tulsiani, S. Song, S. Xu, S. Haldar, S. Karamcheti, S. Adebola, S. Guist, S. Nasiriany, S. Schaal, S. Welker, S. Tian, S. Ramamoorthy, S. Dasari, S. Belkhale, S. Park, S. Nair, S. Mirchandani, T. Osa, T. Gupta, T. Harada, T. Matsushima, T. Xiao, T. Kollar, T. Yu, T. Ding, T. Davchev, T. Z. Zhao, T. Armstrong, T. Darrell, T. Chung, V. Jain, V. Kumar, V. Vanhoucke, V. Guizilini, W. Zhan, W. Zhou, W. Burgard, X. Chen, X. Chen, X. Wang, X. Zhu, X. Geng, X. Liu, X. Liangwei, X. Li, Y. Pang, Y. Lu, Y. J. Ma, Y. Kim, Y. Chebotar, Y. Zhou, Y. Zhu, Y. Wu, Y. Xu, Y. Wang, Y. Bisk, Y. Dou, Y. Cho, Y. Lee, Y. Cui, Y. Cao, Y.-H. Wu, Y. Tang, Y. Zhu, Y. Zhang, Y. Jiang, Y. Li, Y. Li, Y. Iwasawa, Y. Matsuo, Z. Ma, Z. Xu, Z. J. Cui, Z. Zhang, Z. Fu, and Z. Lin. Open X-Embodiment: Robotic learning datasets and RT-X models. https://arxiv.org/abs/2310.08864, 2023.

[25] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. In *RSS*, 2024.

[26] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[27] NVIDIA, :, J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. J. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, J. Jang, Z. Jiang, J. Kautz, K. Kundalia, L. Lao, Z. Li, Z. Lin, K. Lin, G. Liu, E. Llontop, L. Magne, A. Mandlekar, A. Narayan, S. Nasiriany, S. Reed, Y. L. Tan, G. Wang, Z. Wang, J. Wang, Q. Wang, J. Xiang, Y. Xie, Y. Xu, Z. Xu, S. Ye, Z. Yu, A. Zhang, H. Zhang, Y. Zhao, R. Zheng, and Y. Zhu. Gr00t n1: An open foundation model for generalist humanoid robots, 2025. URL https://arxiv.org/abs/2503.14734.

[28] H. Xie, B. Wen, J. Zheng, Z. Chen, F. Hong, H. Diao, and Z. Liu. Dynamicvla: A vision-language-action model for dynamic object manipulation. *arXiv preprint arXiv:2601.22153*, 2026.

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

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

[31] S. Jiang, X. Fang, N. Roy, T. Lozano-Pérez, L. P. Kaelbling, and S. Ancha. Streaming flow policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories, 2025. URL https://arxiv.org/abs/2505.21851.

[32] Y. Lu, Z. Liu, X. Fan, Z. Yang, J. Hou, J. Li, K. Ding, and H. Zhao. Faster: Rethinking real-time flow vlas, 2026. URL https://arxiv.org/abs/2603.19199.

[33] S. Ye, Y. Ge, K. Zheng, S. Gao, S. Yu, G. Kurian, S. Indupuru, Y. L. Tan, C. Zhu, J. Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

[34] M. J. Kim, Y. Gao, T.-Y. Lin, Y.-C. Lin, Y. Ge, G. Lam, P. Liang, S. Song, M.-Y. Liu, C. Finn, et al. Cosmos policy: Fine-tuning video models for visuomotor control and planning. In *ICLR*, 2026.

[35] S. Li, Y. Gao, D. Sadigh, and S. Song. Unified video action model. In *RSS*, 2025.

[36] C. Zhu, R. Yu, S. Feng, B. Burchfiel, P. Shah, and A. Gupta. Unified world models: Coupling video and action diffusion for pretraining on large robotic datasets. In *RSS*, 2025.

[37] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[38] F. Zhang and M. Gienger. Affordance-based robot manipulation with flow matching. *arXiv preprint arXiv:2409.01083*, 2024.

[39] K. Song, B. Chen, M. Simchowitz, Y. Du, R. Tedrake, and V. Sitzmann. History-guided video diffusion. *arXiv preprint arXiv:2502.06764*, 2025.

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

[41] K. Zakka, B. Tabanpour, Q. Liao, M. Haiderbhai, S. Holt, J. Y. Luo, A. Allshire, E. Frey, K. Sreenath, L. A. Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.

[42] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

A Implementation Details

A.1 Simulation

Task. The Leap Cube Reorientation task in MuJoCo Playground [41] requires a 16-DoF Leap Hand to orient a cube without dropping it from the palm. We modify the original environment in two ways: we raise the control rate from 20 Hz to 50 Hz to enable more reactive control, and we restrict the goal distribution to four *blue side up* yaw poses $\{0, \pi/2, \pi, 3\pi/2\}$ rather than uniformly random goals. An episode succeeds if the cube reaches any of the four goals within 0.2 rad in 600 steps; we report the success rate over 100 episodes.

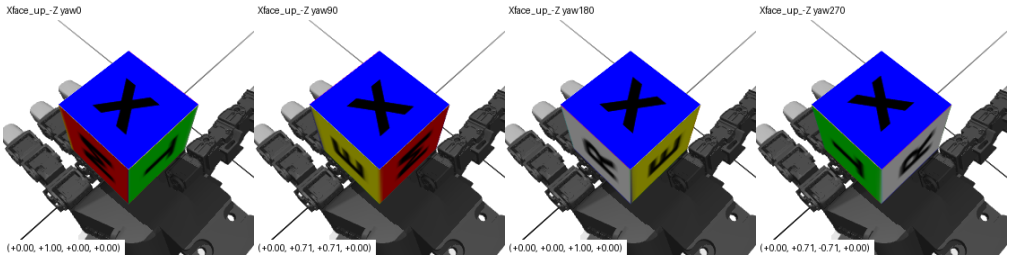


Figure 6: **Leap Cube Reorientation goals.** The Leap Hand grasps a cube and must rotate it until the blue face points up at one of four target yaw poses $(0°, 90°, 180°, 270°)$. The bottom caption shows the corresponding goal quaternions (w, x, y, z).

Dataset. Four PPO experts (different seeds) generate 50 trajectories each, 200 total at 50 Hz. The simulator injects per-step sensing noise into the observation, sampled independently for each component: uniform noise in $[-0.05, 0.05]$ rad on joint angles, uniform noise in $[-0.02, 0.02]$ m on cube position, and Gaussian noise with scale 0.1 on the cube orientation quaternion (the quaternion is re-normalized after perturbation). The recorded demonstrations preserve this noise, so the training distribution already includes realistic sensing noise on both the proprioceptive and vision-derived components.

Observation space. Each observation $\mathbf{o}_t$ concatenates:

- **Proprioception (32 dim)**: noisy joint angles (16) and per-joint tracking error – noisy joint angle minus the most recent motor target (16).
- **Vision-derived cube pose (9 dim)**: palm-to-cube position error (3) and absolute cube orientation in the world frame (6).

The 9-dim vision-derived subset stands in for VLM features per Sec. 3.2; asynchronous-vision/text evaluations apply staleness only to this subset.

Action space. 16-dim relative joint commands at 50 Hz. Each action $\mathbf{a}_t \in [-1, 1]^{16}$ is scaled by 0.5 and added to the current motor target.

Architecture. All variants share a Conditional U-Net 1D action head with chunk length H=16 and n_{obs}=2. The U-Net uses FiLM [42] modulation by the noise level τ at every residual block; $\pi\mathbf{R}^2$ replaces the shared FiLM projection with a per-position one – one (γ_p, β_p) pair per chunk position – analogous to the per-position AdaLN change described in Sec. 3.3 for DiT-based heads. The rest of the architecture is identical across all variants.

Training schedule. At each batch we sample $d \sim \text{Uniform}\{1, \ldots, d_{\max}\}$ with $d_{\max}$=5, build $\boldsymbol{\tau}^{\star,d}$, fill the front-$d$ slots with ground-truth actions, and mask them out of the loss via $m_p = \mathbf{1}[p \geq d]$. With probability 0.2 we instead train on a standard flow schedule (single $\tau \sim \text{Uniform}[0, 1]$ shared across positions, no mask), so the same network can also denoise a full chunk from pure

noise. The standard-flow branch supplies the inference pass that initializes the buffer at episode start (Sec. 3.3, warm-up). For the zero-delay sync curve in Sec. 4.1.1, we train a d=0-only variant ($d_{\max}$=0).

Slow-subset delay in simulation. The simulation policies are purely state-based: no image or language input. The 9-dim vision-derived subset of the state vector (palm-to-cube position error and absolute cube orientation) plays the slow-channel role from Sec. 3.2. In this setting we find that neither delayed-visual-state sampling at training nor the learned delay embedding is necessary: the env already injects sensor noise on this subset at every step, so the policy is already tolerant to perturbed slow inputs out of the box.

For all rows of Sec. 4.1.2, a single $\pi\mathbf{R}^2$ checkpoint is used regardless of d_{vis}: the synchronous slow-channel rows (d_{vis}=0) and the asynchronous-vision/text rows (d_{vis}>0) share the same network, and evaluation simply feeds a d_{vis}-step-old version of the 9-dim subset at test time.

Evaluation protocol. At inference, $\pi\mathbf{R}^2$ performs one denoising step per policy call (each call outputs n_{act} actions). Train-time RTC and standard flow runs K=15 denoising steps to predict the whole chunk. Each (method, d) cell in Sec. 4.1.2 reports the mean success rate over 100 episodes. Full sim hyperparameters are in Tab. 2.

Table 2: **Simulation training and inference hyperparameters** for the Leap Cube Reorientation Conditional U-Net 1D policies. Values shown are $\pi\mathbf{R}^2$ defaults; the right column lists deviations used by baselines. Empty = baseline same as $\pi\mathbf{R}^2$.

Parameter	$\pi\mathbf{R}^2$	Baseline deviations
Shared across all variants		
Chunk length H	16	
Observation history n_{obs}	2	
Control rate	50 Hz	
Optimizer	AdamW (β=[0.95, 0.999])	
Learning rate (peak)	1×10^{-4}	
LR schedule	cosine, 500-step warm-up	
Weight decay	1×10^{-6}	
Gradient-norm clip	1.0	
Batch size (global)	256	
GPUs	1× A5000	
Training budget	800 epochs	
Method-specific (noise schedule + inference)		
Train-time $d_{\max}$	5	10 (Train-time RTC); n/a (Flow)
Standard-flow warm-up prob. α	0.2	n/a
Inference budget per call	1	15 (Train-time RTC, Flow)

Baselines. All three variants share the dataset, optimizer, and training budget. $\pi\mathbf{R}^2$ and train-time RTC additionally share the per-batch delay-sampling protocol: each batch draws $d \in \{0, \dots, d_{\max}\}$ from an exponentially-decaying distribution $p(d{=}k) \propto e^{-\alpha_d k}$ with α_d=1 (smaller d more likely following [30]), and the front-d positions are clamped to ground-truth actions and masked out of the loss. The three variants then differ only in:

- **Flow**: no d sampling and no front clamp; a single shared noise level $\tau \sim \text{Uniform}[0, 1]$ is applied to the entire chunk. This is standard chunked flow matching, evaluated at NFE=15.
- **Train-time RTC** [30]: same per-batch d sampling as $\pi\mathbf{R}^2$, but the noise schedule applies a single shared noise level $\tau \sim \text{Uniform}[0, 1]$ to the back $H{-}d$ positions (no per-position structure). No standard-flow warm-up mixing, and larger $d_{max} = 10$ as the effective delay is larger.
- $\pi\mathbf{R}^2$: same per-batch d sampling as train-time RTC, plus the α=0.2 standard-flow warm-up branch (with probability 0.2 the batch instead uses a single $\tau \sim \text{Uniform}[0, 1]$ on all positions

and no front clamp, so the same network can also initialize a buffer from pure noise at episode start). On the remaining 0.8 of batches we apply the three-region staircase $\boldsymbol{\tau}^{\star,d}$ of Sec. 3.3, and the U-Net uses per-position FiLM heads $\{(\gamma_p, \beta_p)\}_{p=0}^{H-1}$.

A.2 Real-World

Hardware and Workspace. A 6-DoF xArm6 manipulator carries a 12-DoF XHand at its wrist; a single overhead 640×480 RGB camera looks down on the workspace from above (Fig. 7). We fine-tune on $8\times$ NVIDIA RTX A5000/A6000 (24/48 GB VRAM). Deployment uses a single RTX A5000 for the baselines. $\pi\mathbf{R}^2$ uses $2\times$ RTX A5000 so the slow VLM and the fast DiT action head do not share compute or memory bandwidth. Sharing one GPU between the two workers measurably inflates DiT latency and would conflate the measured d.

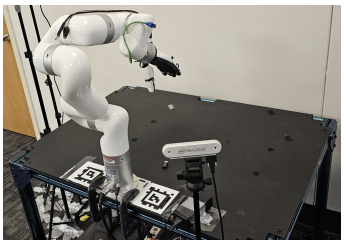


Figure 7: **Real-world workspace.** xArm6 + XHand with a single overhead RGB camera.

Control rate. Predicted actions run closed-loop at 25 Hz at deployment ($T_{\text{ctrl}} \approx 40$ ms per tick). Teleoperation captures demonstrations at the same rate, bottlenecked by the camera and the XHand–workstation USB link.

Base model. We fine-tune NVIDIA GR00T-N1.7 [27]. The Qwen-VL backbone and Eagle-2 vision encoder remain *frozen*; only the action head – state/action projectors, the position embedding, and the DiT – is trained. The only architectural change to the action head replaces the DiT's shared AdaLN modulation with the per-position (γ_p, β_p) heads of Sec. 3.3.

Observation space. Each call to the policy receives:

- **Proprioception** (45 dim): 6 xArm6 joint angles, 12 XHand joint angles (the actively-controlled finger DoFs), 12 corresponding per-joint torques, and 15 fingertip-force values (3-axis force from each of 5 sensorized fingertips).
- **Vision and language**: one 640×480 RGB image from the overhead camera, encoded by Eagle-2 into vision tokens, concatenated with the task prompt (Tab. 4) at the VLM backbone input.

Action space and low-level drivers. Each policy call emits a chunk of $H{=}50$ absolute joint-position targets (6 xArm6 + 12 XHand) at 25 Hz, spanning 2 s of motion. We send one target per control tick to each robot's driver in position-control mode. The arms' onboard firmware interpolates between consecutive commands at their internal servo rates(Tab. 3).

Table 3: **Low-level driver settings.** Both robots run in position-only mode at 25 Hz outbound from the policy; the onboard firmware interpolates between targets at its internal rate.

	xArm6	XHand
Driver mode	`mode=1` (streaming servo)	RS485, position mode
Command call	`set_servo_angle_j`	finger `set_target`
Bus / link	TCP	RS485
Onboard control	API (PID + interp.)	per-joint PID, $k_p{=}100,\ k_i{=}0,\ k_d{=}0$
Safety limits	collision sensitivity $= 2$	torque limit $= 300$

Data collection. We teleoperate the robot setup to collect 200 demonstrations for *Don't Spill*, 300 for *Tidy Up Book* and *Insert Box*, and 100 for *Catch Book*, randomizing the target object's initial pose.

Training the asynchronous vision/text mechanism. The real-world $\pi\mathbf{R}^2$ training explicitly exercises the asynchronous-vision/text mechanism of Sec. 3.2. Each batch samples a vision delay

Table 4: **Real-world task prompts.** Exact language input passed to GR00T-N1.7 at both training and deployment.

Task	Language prompt
Don't Spill	`put the ball in the bowl and put them on the cutting board`
Tidy Up Book	`put the books in the basket`
Insert Box	`put the box in the basket`
Catch Book	`catch the book`

$d_{\text{vis}} \sim \text{Uniform}\{0, \ldots, d_{\text{vis}}^{\max}\}$ with $d_{\text{vis}}^{\max}$=5 ticks (200 ms at 25 Hz); the policy then receives the image recorded d_{vis} ticks earlier (i.e., the corresponding past frame in the demonstration), while proprioception stays current. For each batch, the integer $d_{\text{vis}} \in \{0, \ldots, d_{\text{vis}}^{\max}\}$ indexes a learned embedding $e(d_{\text{vis}})$ from a $(d_{\text{vis}}^{\max}+1)$-entry lookup table. The DiT adds $e(d_{\text{vis}})$ to its action-token features (broadcast across the H chunk positions), so the action head conditions on both the cached visual feature and its age. We zero-initialize the lookup table so that the untrained checkpoint reproduces the no-delay variant exactly. At deployment, the measured wall-clock vision latency passes through the same embedding before each DiT call.

Training hyperparameters. We fine-tune the GR00T-N1.7 action head (projectors + DiT) and keep the VLM backbone and vision encoder frozen. Full optimizer settings, batch size, and step budgets are in Tab. 5. The per-task budgets (40,000 steps for *Don't Spill*, 28,000 for *Tidy Up Book*, 49,000 for *Insert Box*, 4, 550 for *Catch Book*) correspond to approximately $200/100/100/100$ epochs over each dataset. The per-position AdaLN parameters $\{(\gamma_p, \beta_p)\}_{p=0}^{H-1}$ initialize from the pretrained shared pair, so each head starts from a position-uniform schedule and gradually specializes during fine-tuning. All four rows in Tab. 1 fine-tune from the same GR00T-N1.7 checkpoint with identical data and budget; they differ only in the noise schedule and (for $\pi\mathbf{R}^2$) the AdaLN modification. The train-time RTC baseline samples its per-batch delay from $d \in \{0, \ldots, 10\}$ (i.e., $d_{\max}$=10) rather than the $d_{\max}$=5 used in $\pi\mathbf{R}^2$, as $\pi\mathbf{R}^2$is separating the delay into two parts while RTC treats it as a whole. Full hyperparameters are in Tab. 5.

Evaluation protocol. Each $(\text{method}, \text{task})$ cell runs N=20 trials with randomized initial object placement. Success rate (SR) counts whole-task completion within the episode time limit (30 seconds). The progress score (Prog) decomposes each task into sub-goals and reports the per-trial fraction completed:

- *Don't Spill* – 4 sub-goals: pick up ball, place ball in bowl, lift bowl off the table, place bowl on the cutting board.
- *Tidy Up Book* – 2 sub-goals: pull a book free from the pile, place it inside the basket.
- *Insert Box* – 4 sub-goals: push the box towards the wall, make it stand, grasp, and insert between books.
- *Catch Book* – 1 sub-goal: grasp the falling book and pick it up.

Deployment query modes. The four rows of Tab. 1 differ in how the action-head worker queries the policy:

- *Flow, Synchronous*: the main loop blocks until each call returns; the robot freezes during inference.
- *Flow, Naive Async, dense, ensemble*: the action worker queries back-to-back and combines overlapping chunks with the temporal-ensembling weights of Zhao et al. [16].
- *Train-time RTC* and $\pi\mathbf{R}^2$: the action worker queries back-to-back; the new chunk replaces the active one at the next swap boundary. The actions executed during policy call is given as input. $\pi\mathbf{R}^2$ additionally spawns a VLM-cache worker on a separate GPU that refreshes the cached visual features in the background.

Table 5: **Real-world training and inference hyperparameters** for the GR00T-N1.7 fine-tunes. Values shown are $\pi\mathbf{R}^2$ defaults; the right column lists deviations used by baselines. Empty = baseline same as $\pi\mathbf{R}^2$.

Parameter	$\pi\mathbf{R}^2$	Baseline deviations
Shared across all variants		
Chunk length H	50	
Observation history n_{obs}	1	
Control rate	25 Hz	
Optimizer	fused AdamW	
Learning rate (peak)	1×10^{-4}	
LR schedule	cosine, 5% warm-up	
Weight decay	1×10^{-5}	
Gradient-norm clip	1.0	
Batch size	512	
Precision	bf16 + tf32	
GPUs	8× A5000/A6000	
Training budget	100 epoch (*Tidy Up Book* / *Insert Box* *Catch Book*) / 200 epoch (*Don't Spill*)	
Method-specific (noise schedule + delay + inference)		
Train-time $d_{\max}$	5	10 (Train-time RTC); n/a (Flow)
Train-time $d_{\text{vis}}^{\max}$	5	n/a
Delay-embedding lookup size	6 entries → DiT hidden dim	n/a
Standard-flow warm-up prob. α	0.2	0 (Flow, Train-time RTC)
NFE per call	1 (one sub-chunk)	4 (Flow, Train-time RTC; full chunk)

A.3 Training and Inference Algorithms

We give pseudocode for the per-batch training step (Alg. 1) and the deployment-time inference loop (Alg. 2). The inference loop is the loop implemented in our deployment script; the wall-clock measurement that auto-derives d uses a rolling window of the most recent query times.

Algorithm 1 $\pi\mathbf{R}^2$ training step. Each batch samples a delay d, builds the staircase $\boldsymbol{\tau}^{\star,d}$, masks the front, applies jitter, and (for real-world training) additionally delays the slow channel.

Require: Action chunk $\mathbf{a}_{0:H-1}$, observation $(\mathbf{o}_t^{\text{fast}}, \mathbf{o}_{t:t-T_{\text{slow}}+1}^{\text{slow}})$, model v_θ
Require: Hyperparams $d_{\max}, d_{\text{vis}}^{\max}, j, \alpha$
$u \sim \text{Uniform}[0,1]$
if $u < \alpha$ **then** ▷ warm-up branch: standard flow
  $t \sim \text{Uniform}[0,1]$; $\tau_p \leftarrow t,\ \forall p$
  $\mathbf{x}_{\tau,p} \leftarrow (1-\tau_p)\boldsymbol{\epsilon}_p + \tau_p \mathbf{a}_p$
  $m_p \leftarrow 1,\ \forall p$ ▷ all positions contribute to loss
else ▷ staircase branch
  $d \sim \text{Uniform}\{1,\dots,d_{\max}\}$
  Build $\boldsymbol{\tau}^{\star,d}$ as in Sec. 3.3 ▷ three-region staircase
  $\delta_p \sim \text{Uniform}[-j,j]$; $\tau_p \leftarrow \text{clip}(\tau_p^{\star,d} + \delta_p, 0, 1)$
  $\mathbf{x}_{\tau,p} \leftarrow (1-\tau_p)\boldsymbol{\epsilon}_p + \tau_p \mathbf{a}_p$
  $\mathbf{x}_{\tau,p} \leftarrow \mathbf{a}_p$ if $p < d$ ▷ front d slots are ground-truth (inpaint conditioning)
  $m_p \leftarrow \mathbf{1}[p \geq d]$
end if
$d_{\text{vis}} \sim \text{Uniform}\{0,\dots,d_{\text{vis}}^{\max}\}$ ▷ slow-channel staleness (real-world only)
$\mathbf{o}^{\text{slow}} \leftarrow \mathbf{o}_{t-d_{\text{vis}}}^{\text{slow}}$; append $e(d_{\text{vis}})$ to slow representation
$\hat{\mathbf{v}}_p \leftarrow v_\theta(\mathbf{x}_\tau, \boldsymbol{\tau}, \mathbf{o}_t^{\text{fast}}, \mathbf{o}^{\text{slow}})_p$
$\mathcal{L} \leftarrow \sum_{p=0}^{H-1} m_p \,\|\hat{\mathbf{v}}_p - (\mathbf{a}_p - \boldsymbol{\epsilon}_p)\|^2$
Update θ on $\mathcal{L}$

Algorithm 2 $\pi\mathbf{R}^2$ inference loop on the robot. The action head runs continuously with fresh proprioception and a cached slow feature; the VLM runs asynchronously in a background thread; the per-call delay d is auto-derived from a rolling measurement of action-head wall-clock latency.

Require: Policy v_θ, observation source, robot controller, control period T_{ctrl}
Require: Rolling window W of recent query times (e.g. $W{=}20$)
$\mathbf{C} \leftarrow$ `WarmStart()` ▷ Sec. 3.3: full standard-flow inference, denoise all H positions
$\mathbf{C} \leftarrow$ re-noise $\mathbf{C}$ to match $\boldsymbol{\tau}^{\star,d_0}$ for initial estimate d_0
$i \leftarrow 0$ ▷ chunk index
queue $Q \leftarrow \emptyset$ ▷ rolling latency window
Spawn `VLM_Worker` thread (continuously updates cached slow feature)
Spawn `Action_Worker` thread (continuously queries action head)
loop ▷ one iteration per control tick (25 Hz)
  $t_0 \leftarrow$ current wall-clock time
  Read fresh proprioception $\mathbf{o}_t^{\text{fast}}$ from robot sensors
  Publish $\mathbf{o}_t^{\text{fast}}$ to shared state for workers
  if new chunk $\mathbf{C}_{\text{new}}$ is available from `Action_Worker` **then**
    $\mathbf{C} \leftarrow \mathbf{C}_{\text{new}}$; $i \leftarrow d$ ▷ swap and skip past in-flight front actions
  end if
  Send action $\mathbf{C}_i$ to robot; $i \leftarrow i + 1$
  Sleep until $t_0 + T_{\text{ctrl}}$
end loop

procedure ACTION_WORKER ▷ background thread; one denoising step per call
  loop
    Snapshot $(\mathbf{o}_t^{\text{fast}}, t_{\text{state}})$ and the cached $(\mathbf{o}_{\text{cached}}^{\text{slow}}, t_{\text{image}})$
    $d_{\text{vis}} \leftarrow \text{round}\big((t_{\text{state}} - t_{\text{image}})/T_{\text{ctrl}}\big)$ ▷ age of cached slow feature in control ticks
    $d \leftarrow \max\big(1,\ \text{round}(\text{mean}(Q)/T_{\text{ctrl}})\big)$ ▷ auto-derived from rolling window
    Snapshot front-d inpaint: $\mathbf{a}_{0:d}^{\text{inflight}} \leftarrow \mathbf{C}_{i:i+d}$
    $q_0 \leftarrow$ wall-clock
    $\mathbf{C}_{\text{new}} \leftarrow v_\theta$`.euler_step`$(\mathbf{a}^{\text{inflight}}, \mathbf{o}^{\text{fast}}, \mathbf{o}_{\text{cached}}^{\text{slow}}, e(d_{\text{vis}}))$ ▷ one NFE; per-position $\Delta\tau_p$ as in Eq. (4)
    Push wall-clock $- q_0$ into Q
    Hand off $\mathbf{C}_{\text{new}}$ to main loop
  end loop
end procedure

procedure VLM_WORKER ▷ background thread; runs vision/text forward asynchronously
  loop
    Read fresh image and language; $t_{\text{image}} \leftarrow$ image capture time
    $\mathbf{o}_{\text{cached}}^{\text{slow}} \leftarrow$ VLM forward (image, language)
    Atomically update cache with $(\mathbf{o}_{\text{cached}}^{\text{slow}}, t_{\text{image}})$
  end loop
end procedure

A.4 Additional Reactivity Analysis

The main-text reactivity comparison (Fig. 5) is on *Tidy Up Book*; the same pattern holds across the other tasks (Fig. 8). On *Catch Book*, $\pi\mathbf{R}^2$ closes on the force spike as the book lands, whereas Train-Time RTC reacts too late and the book slips. On *Insert Box*, RTC reacts late to the force feedback and its index force overshoots out of distribution, while $\pi\mathbf{R}^2$ stays controlled. On *Don't Spill*, RTC's thumb never contacts the ball (thumb force ≈ 0) yet it keeps going, while $\pi\mathbf{R}^2$ grasps it with both fingers.

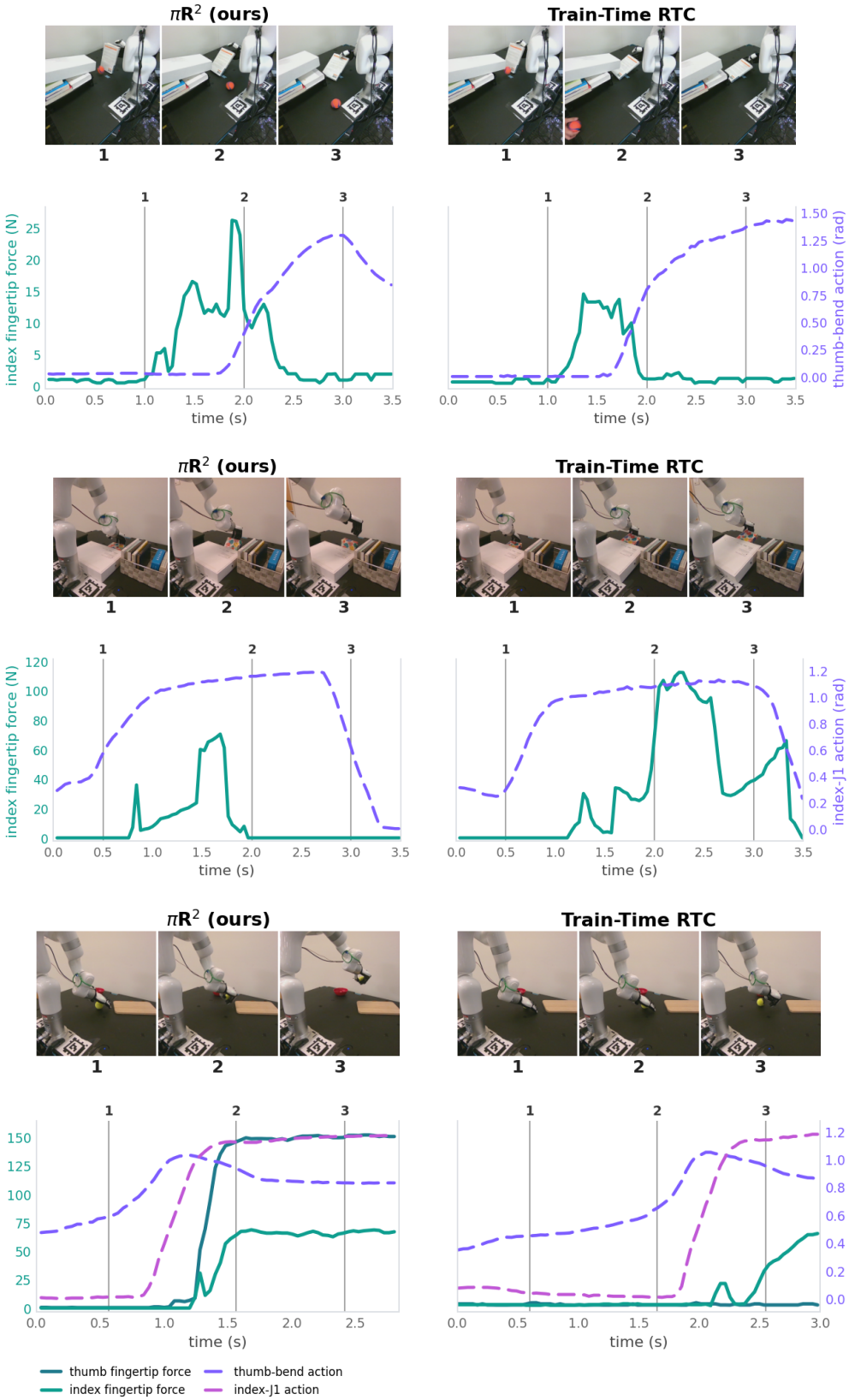


Figure 8: **Reactivity on the remaining tasks.** Fingertip force (solid, left axis) and the emitted action (dashed, right axis) over time for $\pi\mathbf{R}^2$ and Train-Time RTC, with overhead frames at the marked times. Top: *Catch Book*. Middle: *Insert Box*. Bottom: *Don't Spill*.