$\pi\mathbf{R}^2$: REACTIVE REAL-TIME FLOW POLICIES

Anonymous authors
Paper under double-blind review

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, $\pi\mathbf{R}^2$ maintains a partially denoised action buffer and advances it under deployment latency. Conditioning is split 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; a latency-adaptive flow schedule then treats the 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, acting on a fresh observation every 40 ms (25 Hz). Across simulation and real-world manipulation tasks, $\pi\mathbf{R}^2$ improves the success rate by up to 25% in simulation and 30% in the real world over the strongest baseline.

Project page:https://pir2-iclr2027.github.io/

1 INTRODUCTION

We have witnessed remarkable progress in training 'robotics foundation models' (Black et al., 2025a; Intelligence et al., 2025; Bjorck et al., 2025; Brohan et al., 2023; Zitkovich et al., 2023; Kim et al., 2024b), and three design choices have become ubiquitous across recent efforts. First, **large pre-trained backbones** such as vision-language models (VLMs) (Liu et al., 2023; Alayrac et al., 2022; Bai et al., 2025; Team et al., 2023; Beyer et al., 2024; Deitke et al., 2024) are leveraged to process visual (and language) input and extract rich, generalizable representations for downstream action prediction. Second, **expressive policy architectures** such as diffusion (Chi et al., 2023) and flow matching (Yuan et al., 2024; Pan et al., 2025) allow effectively capturing the multi-modal action distributions inherent in diverse human demonstrations. Finally, **action chunking** (Zhao et al., 2023; Zhang et al., 2025) 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 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:

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

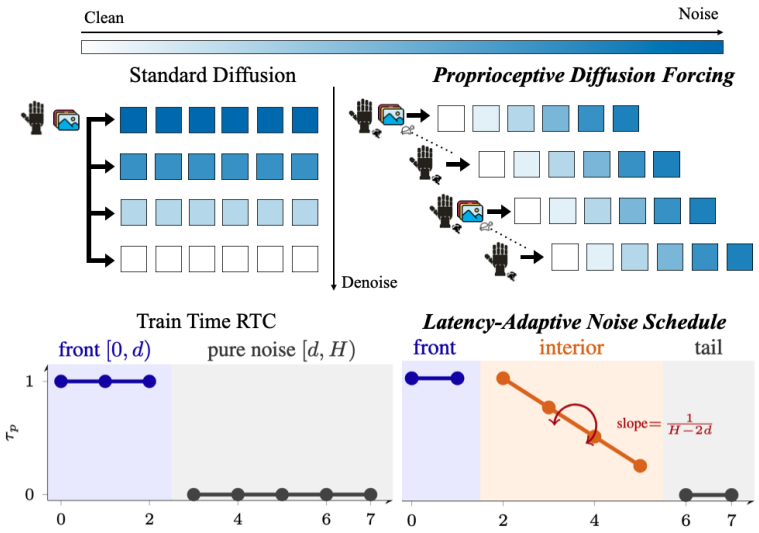


Figure 1: Overview of $\pi\mathbf{R}^2$. **Top.** While standard diffusion/flow matching relies on stale observations to predict actions via iterative denoising, $\pi\mathbf{R}^2$ disentangles the observation into a fast channel (proprioception) and a slow channel (image encoding, VLM embedding, etc.), 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 (Chen et al., 2024), and pure noise with the same length as clean actions. Compared to Train-Time RTC, this enables faster inference and smoother actions.

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' Chen et al. (2024) 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 Høeg et al. (2024), 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, even with asynchronous perception, practical constraints (communication delays, GPU bandwidth) prevent zero latency, so some actions are always already in flight when the next call begins. 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 (Bjorck et al., 2025)). 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 25% 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 (Zhao et al., 2023), improves the temporal coherence of imitation-learned policies and has since become standard, including for mobile and contact-rich manipulation (Fu et al., 2024; Zhao et al., 2024). This design has been widely adopted by expressive multi-modal policies based on diffusion (Chi et al., 2023) and flow matching (Zhang et al., 2024; Yan et al., 2025). 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 (Collaboration et al., 2023; Khazatsky et al., 2024; Zitkovich et al., 2023; Kim et al., 2024a). 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 (Black et al., 2025a; Intelligence et al., 2025; Bjorck et al., 2025). 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 (Xie et al., 2026). 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 (Black et al., 2026; 2025b). Streaming Diffusion Policy and Streaming Flow Policy emit actions incrementally from a diffusion-forcing-style variable-noise buffer maintained across observations (Høeg et al., 2024; Jiang et al., 2025). 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 (Lu et al., 2026) 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 (Ye et al., 2026; Kim et al., 2026; Li et al., 2025; Zhu et al., 2025); 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 (Lipman et al., 2022; Zhang & Gienger, 2024) with action chunking (Zhao et al., 2023; Zhang et al., 2025). 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 $h \le H$ 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 (Chen et al., 2024; Song et al., 2025) 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* (Høeg et al., 2024) 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 an RTX A5000/A6000 with GR00T-N1.7, image preprocessing + VLM ($\sim$ 60 ms) plus DiT denoising (by default K=4 steps, $\sim$ 80 ms) sum to $\sim$ 140 ms per call, $\sim$ 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 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.

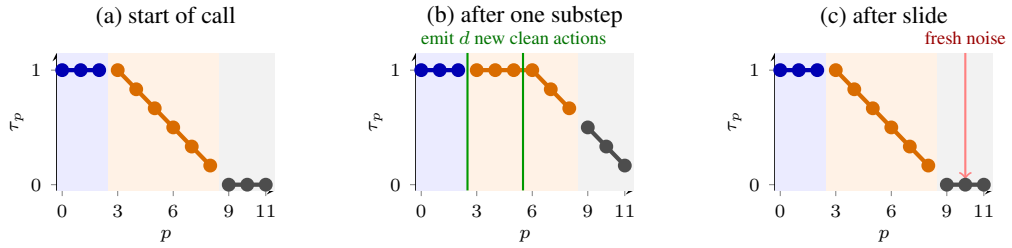


Figure 2: **Inference cycle, one call.** (a) Buffer at $\tau^{\star,d}$ at the start of the call. (b) One Euler substep (one NFE) shifts the schedule right by d slots: positions $[d, 2d)$ reach $\tau{=}1$ and are released, and the ramp extends through the back (c) The buffer slides d positions: just-emitted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

To absorb the resulting slow-channel staleness, we delay the slow channel at training time by $d_{\text{vlm}} \in \{0, \dots, d_{\text{vlm}}^{\max}\}$. The resulting policy is *proprioception-reactive*: it reacts to fresh joint state every denoising step, recovering closed-loop control even behind a large VLA backbone. Furthermore, a shorter per-call latency permits a shorter execution horizon, so the slow features behind the actions actually *executed* are on average *fresher* than a synchronous baseline's (120 vs 210 ms at our measured costs; App. A.3). 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 (Høeg et al., 2024), 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 (Black et al., 2026; Tang et al., 2025). 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 $\boldsymbol{\tau}^{\star,d} \in [0,1]^H$

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

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 (Black et al., 2025b): 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 a delay d between $\{0, \dots, d_{\max}\}$ and build $\boldsymbol{\tau}^{\star,d}$. The front d slots sit at $\tau{=}1$ and therefore equal the ground-truth actions; the interior and tail are noised at the staircase levels and contribute to the per-position MSE. We additionally apply a symmetric jitter

$\tau_p \leftarrow \text{clip}(\tau_p + \delta, 0, 1)$, where a single $\delta \sim \text{Uniform}[-j, j]$ with $j = \frac{1}{4(H-2d)}$ is drawn per training example and shared across positions, to absorb small deviations from the central staircase that arise from per-call d variation. After the jitter, the loss mask is applied on the resulting noise level: $m_p = \mathbf{1}[\tau_p < 1]$. With probability $p = 0.2$, 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, 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, pre-trained 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 test $\pi\mathbf{R}^2$ on the Leap Cube Reorientation task in MuJoCo Playground (Zakka et al., 2025), a common task that requires reactivity to keep the cube from falling and to achieve the target cube pose. Demonstrations are collected 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 realistic 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 15 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(a). Intuitively, the performance of standard flow degrades as h grows, confirming the h tradeoff for reactive tasks. Meanwhile, $\pi\mathbf{R}^2$ (0.93 ± 0.01) matches standard flow at $h{=}2$ (0.93 ± 0.02) and clearly exceeds $h{\geq}4$, while falling modestly short of $h{=}1$ (0.96 ± 0.02). This supports that leveraging fresh observation more frequently is beneficial for such reactive tasks, and that amortizing the denoising budget across calls costs little 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 $\approx$ 60 ms, $K{=}4$ NFE denoising $\approx$ 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 denoising $= d_0/4$.

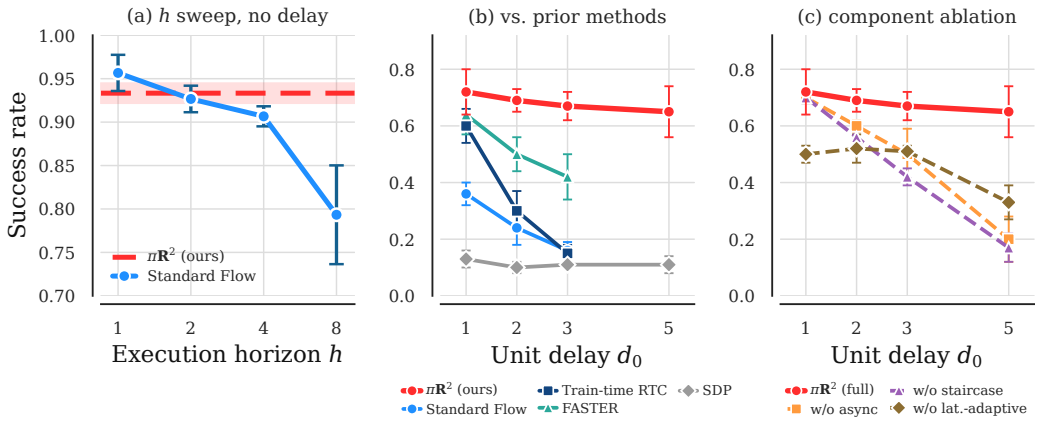


Figure 3: **Simulation results on Leap Cube Reorientation. (a)** Without inference delay, executing fewer actions per chunk benefits a flow matching policy; $\pi\mathbf{R}^2$ matches this by replanning every timestep, with only the last denoising step conditioned on the up-to-date state. **(b,c)** Success rate as the unit delay d_0 grows; error bars show std over 3 training seeds. Each method is charged the delay its own deployment incurs (Sec. 4.1.2); at d_0=5 that exceeds the $d \leq H/2 = 8$ the schedule can accommodate for Standard Flow, Train-time RTC and FASTER. **(c)** Removing any single component costs performance, and the gap widens with d_0.

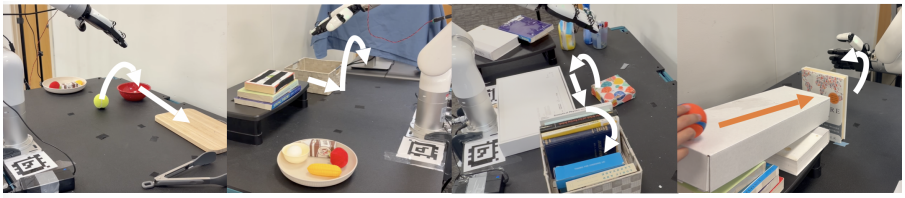
The simulation policies are state-based, with a 9-dim vision-derived subset of the state playing the slow-channel role (Appendix A).

Baselines. We compare against four prior methods. (i) *Standard Flow (async)* predicts the new chunk while executing the previous one, without accounting for the in-flight actions. (ii) *Train-time RTC* (Black et al., 2025b) conditions on those in-flight actions, giving smooth chunk boundaries. (iii) *FASTER* (Lu et al., 2026) reinitializes the chunk to pure noise every call and needs gradually more denoising steps to emit actions for further future, so only the first action can be denoised in one step. (iv) *SDP* (Høeg et al., 2024) applies a staircase schedule at fixed d=0, assuming zero inference delay, so under real latency the robot must stall. Moreover, all baselines do not decouple slow/fast features.

Each method is charged the delay its *own* deployment actually incurs. *Standard Flow* and *Train-time RTC* run the full pipeline serially and pay $d = \lceil 1.75\, d_0 \rceil$. *FASTER* releases actions progressively, so we charge it the ready-step of the last position it emits under its own schedule (α=0.6, u_0=0.9), giving $d = 1, 3, 4$ at $d_0 = 1, 2, 3$. *SDP* requires one denoising step per call and assumes zero inference delay, so it pays $d = d_0$, and stalls between inferences. $\pi\mathbf{R}^2$ pays $d = \lceil 0.25\, d_0 \rceil$ and additionally delays the object pose by $d_{\text{vis}} = \lceil 0.75\, d_0 \rceil$; the w/o-async ablation pays $d = \lceil 1.0\, d_0 \rceil$. We set the execution horizon $h = d$ throughout, and set $d_0 \in \{1, 2, 3, 5\}$. For continuous execution, the schedule must fit a clean prefix and a noise tail of length d each inside the chunk, so it requires $d \leq H/2 = 8$; at d_0=5 this excludes Standard Flow, Train-time RTC and FASTER, while SDP (d=5, stalls in between) and $\pi\mathbf{R}^2$ (d=2) still fit.

Comparison to baselines. Fig. 3(b) reports the mean over three training seeds ($\pm$ std), picking the best epoch per seed. $\pi\mathbf{R}^2$ wins at every d_0 (0.72, 0.69, 0.67, 0.65), and its margin widens sharply with delay: against FASTER, the strongest prior method, the absolute gap grows from 8 points at d_0=1 to 25 at d_0=3. Train-time RTC is competitive at the shortest delay (0.60) but collapses to 0.15 by d_0=3, and standard async flow degrades from 0.36 to 0.16. SDP stays weak ($\approx$ 0.11–0.13), as it assumes zero delay and must stall. The cause of this ordering is reducing effective delay.

Ablations. We additionally ablate individual components of $\pi\mathbf{R}^2$, shown in Fig. 3. *W/o async slow channel* refreshes the slow features synchronously with the action head, so the per-call delay reverts to the full backbone cost with one denoising step. *W/o staircase* keeps the asynchronous split but replaces the ramped schedule with train-time RTC's single shared noise level over the non-clamped positions, requiring multiple denoising steps. It is basically applying asynchronous slow feature processing on top of train-time RTC. *W/o latency-adaptive* trains with $d_{\max}$=0, so the



***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.

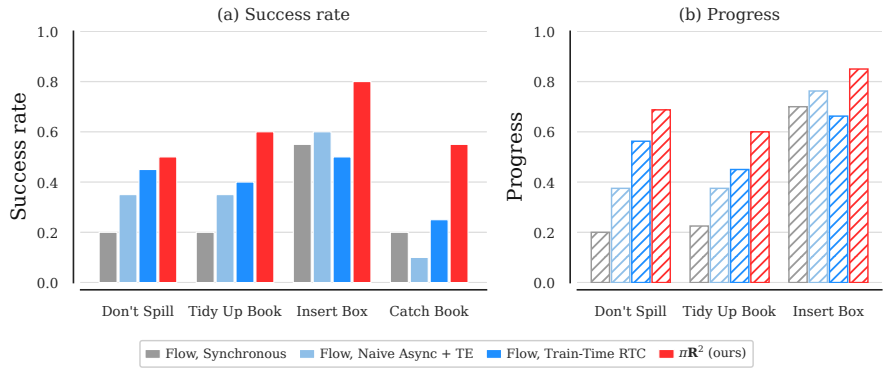


Figure 5: **Real-world results** (xArm6 + XHand, fine-tuned GR00T-N1.7). **(a)** Success rate over N=20 trials per task. **(b)** Progress score, the fraction of subgoals completed per episode; *Catch Book* is single-stage and has no progress metric, so it appears in (a) only.

schedule cannot condition on the delay and the robot must stall while a chunk is in flight. At d_0=3, success falls from 0.67 for the full method to 0.50 without the async slow channel, 0.42 without the staircase, and 0.51 without the latency-adaptive noise schedule; by d_0=5 the first two collapse to 0.20 and 0.17. This supports the necessity of combining individual components proposed in $\pi\mathbf{R}^2$.

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 (Bjorck et al., 2025) 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). This yields $d = 4 \sim 5$ 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 both visual and proprioceptive reactivity, needing vision to track the fall and contact feedback to secure the grasp. 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 (execution horizon $h = 10$). (2) *Flow, Naive Async (dense, TE)* (Zhao et al., 2024) 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) (Zhao et al., 2023) for overlapping action predictions for the same timestep. (3) *Flow,*

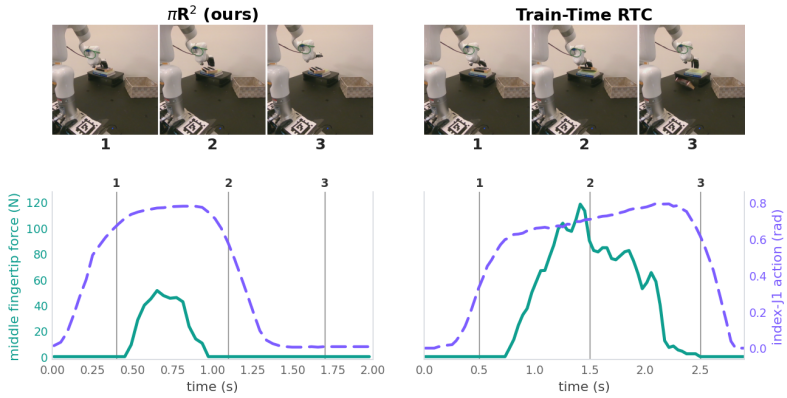


Figure 6: $\pi\mathbf{R}^2$ **reacts to proprioception, while baselines run a stale plan** (*Tidy Up Book*). Fingertip force (solid, left axis) and the emitted action (dashed, right axis) over time, for $\pi\mathbf{R}^2$ and Train-Time RTC; numbered markers link plot times to the overhead frames above. $\pi\mathbf{R}^2$ grips just enough ($\sim$50 N), while RTC reacts late and over-grips to $\sim$120 N, dropping the book.

Train-Time RTC (Black et al., 2025b) additionally trains the policy to inpaint a clean d-action front, anchoring the new chunk to the executed history at the chunk boundary.

Results. Fig. 5 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 (Black et al., 2025b) is the strongest baseline; however, $\pi\mathbf{R}^2$ outperforms it across all metrics, with the largest gains on reactivity-critical tasks such as Tidy Up Book, Insert Box, and Catch Book, achieving $20 \sim 30\%$ 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, as shown in the website. These results demonstrate the value of the real-time proprioceptive feedback enabled by $\pi\mathbf{R}^2$.

Reactivity analysis. Fig. 6 plots a fingertip contact force against the action each method emits over time on *Tidy Up Book*. As $\pi\mathbf{R}^2$ 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. The same pattern holds in all remaining tasks (Appendix A.5).

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

AI USE STATEMENT

In this work, we used generative AI tools primarily to rephrase and polish portions of the manuscript for clarity and readability. The authors designed and conducted the main experiments, and analyzed the results. We reviewed the AI-assisted edits to ensure that the final text accurately represents our methods and results. We did not use generative AI for data generation or assisting mathematical claims. The authors take responsibility for the entire submission.

ETHICS STATEMENT

This work studies real-time control for robot manipulation and does not involve human subjects, personally identifying information, or sensitive data. All demonstrations were collected by the authors teleoperating a robot in a controlled laboratory workspace, and all evaluations were run either in simulation or on that same hardware. The primary safety consideration is physical: $\pi\mathbf{R}^2$ increases the rate at which a manipulator commits to new actions, which raises the importance of conventional workspace safeguards (force limits, workspace bounds, and an accessible emergency stop), all of which were in place during our experiments. We do not release any dataset containing third-party data. More broadly, methods that make large manipulation policies reactive enough for contact-rich deployment lower the barrier to operating such systems around people, and we encourage downstream users to pair them with the safety interlocks described above.

REPRODUCIBILITY STATEMENT

We have made the following efforts to support reproducibility. The method is specified in Sec. 3, with the training and inference procedures given as explicit pseudocode in Appendix A.4. Appendix A documents the full experimental setup for both domains: for simulation, the task, demonstration collection, observation and action spaces, architecture, training schedule, and the evaluation protocol each method's effective delay d follows from the unit delay d_0 as given in Sec. 4.1.2 for the baselines and App. A for the ablations; for the real-world experiments, the hardware and workspace, control rate, base model, observation and action spaces, low-level drivers, data collection procedure, training hyperparameters, and evaluation protocol. Simulation results are reported as the mean and standard deviation over 3 training seeds. For each seed we take the best checkpoint over a pool matched across methods within each study. The simulation environment is the publicly available Leap Cube Reorientation task from MuJoCo Playground (Zakka et al., 2025), and our real-world experiments fine-tune the publicly released GR00T-N1.7 (Bjorck et al., 2025). We also upload anonymized code for our real-world experiments.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bošnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier Henaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. Paligemma: A versatile 3b vlm for transfer, 2024. URL `https://arxiv.org/abs/2407.07726`.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. $\pi 0$: A vision-language-action flow model for general robot control. In *RSS*, 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.

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38:33383–33407, 2026.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. In *RSS*, 2023.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *IJRR*, 2023.

Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Felipe Vieira Frujeri, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guangwen Yang, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homanga Bharadhwaj, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jay Vakil, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi "Jim" Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minho Heo, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Muhammad Zubair Irshad, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, Ning Liu, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R Sanketi, Patrick "Tree" Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundaresan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Mart'in-Mart'in, Rohan Baijal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shubham Tulsiani, Shuran Song, Sichun Xu, Siddhant Haldar, Siddharth Karamcheti, Simeon

Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vikash Kumar, Vincent Vanhoucke, Vitor Guizilini, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiangyu Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yansong Pang, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yongqiang Dou, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen Zhang, Zipeng Fu, and Zipeng Lin. Open X-Embodiment: Robotic learning datasets and RT-X models. https://arxiv.org/abs/2310.08864, 2023.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, YenSung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, Andrew Head, Rose Hendrix, Favyen Bastani, Eli VanderBilt, Nathan Lambert, Yvonne Chou, Arnavi Chheda, Jenna Sparks, Sam Skjonsberg, Michael Schmitz, Aaron Sarnat, Byron Bischoff, Pete Walsh, Chris Newell, Piper Wolters, Tanmay Gupta, Kuo-Hao Zeng, Jon Borchardt, Dirk Groeneveld, Crystal Nam, Sophie Lebrecht, Caitlin Wittlif, Carissa Schoenick, Oscar Michel, Ranjay Krishna, Luca Weihs, Noah A. Smith, Hannaneh Hajishirzi, Ross Girshick, Ali Farhadi, and Aniruddha 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.

Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation, 2024. URL https://arxiv.org/abs/2401.02117.

Sigmund H Høeg, Yilun Du, and Olav Egeland. Streaming diffusion policy: Fast policy synthesis with variable noise diffusion models. *arXiv preprint arXiv:2406.04806*, 2024.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

Sunshine Jiang, Xiaolin Fang, Nicholas Roy, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Siddharth Ancha. Streaming flow policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories, 2025. URL https://arxiv.org/abs/2505.21851.

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. In *RSS*, 2024.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024a.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. In *CoRL*, 2024b.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, et al. Cosmos policy: Fine-tuning video models for visuomotor control and planning. In *ICLR*, 2026.

Shuang Li, Yihuai Gao, Dorsa Sadigh, and Shuran Song. Unified video action model. In *RSS*, 2025.

Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.

Yuxiang Lu, Zhe Liu, Xianzhe Fan, Zhenya Yang, Jinghua Hou, Junyi Li, Kaixin Ding, and Hengshuang Zhao. Faster: Rethinking real-time flow vlas, 2026. URL https://arxiv.org/abs/2603.19199.

Chaoyi Pan, Giri Anantharaman, Nai-Chieh Huang, Claire Jin, Daniel Pfrommer, Chenyang Yuan, Frank Permenter, Guannan Qu, Nicholas Boffi, Guanya Shi, et al. Much ado about noising: Dispelling the myths of generative robotic control. *arXiv preprint arXiv:2512.01809*, 2025.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Kiwhan Song, Boyuan Chen, Max Simchowitz, Yilun Du, Russ Tedrake, and Vincent Sitzmann. History-guided video diffusion. *arXiv preprint arXiv:2502.06764*, 2025.

Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Haozhe Xie, Beichen Wen, Jiarui Zheng, Zhaoxi Chen, Fangzhou Hong, Haiwen Diao, and Ziwei Liu. Dynamicvla: A vision-language-action model for dynamic object manipulation. *arXiv preprint arXiv:2601.22153*, 2026.

Ge Yan, Jiyue Zhu, Yuquan Deng, Shiqi Yang, Ri-Zhao Qiu, Xuxin Cheng, Marius Memmel, Ranjay Krishna, Ankit Goyal, Xiaolong Wang, and Dieter Fox. Maniflow: A general robot manipulation policy via consistency flow training, 2025. URL https://arxiv.org/abs/2509.01819.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

Chengbo Yuan, Chuan Wen, Tong Zhang, and Yang Gao. General flow as foundation affordance for scalable robot learning. *arXiv preprint arXiv:2401.11439*, 2024.

Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.

Fan Zhang and Michael Gienger. Affordance-based robot manipulation with flow matching. *arXiv preprint arXiv:2409.01083*, 2024.

Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng 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.

Thomas T Zhang, Daniel Pfrommer, Chaoyi Pan, Nikolai Matni, and Max Simchowitz. Action chunking and exploratory data collection yield exponential improvements in behavior cloning for continuous control. *arXiv preprint arXiv:2507.09061*, 2025.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Tony Z Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Kamyar Ghasemipour, Chelsea Finn, and Ayzaan Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024.

Chuning Zhu, Raymond Yu, Siyuan Feng, Benjamin Burchfiel, Paarth Shah, and Abhishek Gupta. Unified world models: Coupling video and action diffusion for pretraining on large robotic datasets. In *RSS*, 2025.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *CoRL*, 2023.

A Implementation Details

A.1 Simulation

Task. The Leap Cube Reorientation task in MuJoCo Playground (Zakka et al., 2025) requires a 16-DoF Leap Hand to orient a cube without dropping it from the palm. We keep the environment's native 20 Hz control rate and make one modification: we restrict the goal distribution to four *blue side up* yaw poses $\{0, \pi/2, \pi, 3\pi/2\}$ rather than uniformly random goals. The discrete goal set to follow the usual VLA setup, where goals are given through language and a continuous SE(3) target is not expressible as an instruction. An episode succeeds if the cube reaches any of the four goals within 0.2 rad in 360 steps (18 s at 20 Hz); we report the success rate over 100 episodes.

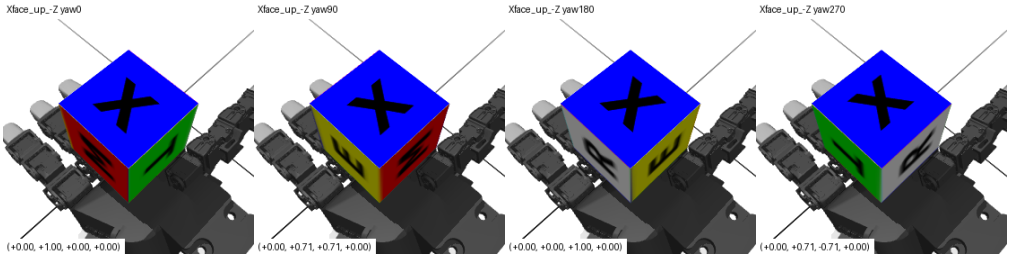


Figure 7: **Leap Cube Reorientation goals.** The Leap Hand grasps a cube and must rotate it until the blue face points up at one of four target yaw poses $(0°, 90°, 180°, 270°)$. The bottom caption shows the corresponding goal quaternions (w, x, y, z).

Dataset. Four PPO experts (different seeds) generate 50 trajectories each, 200 total at 20 Hz, each truncated to its first 360 steps. 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 20 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 (Perez et al., 2018) modulation by the noise level τ at every residual block; $\pi\mathbf{R}^2$ evaluates that same FiLM projection at each chunk position's own noise level τ_p (implemented as a 1×1 convolution over the chunk axis, so the projection weights stay shared), giving a distinct (γ_p, β_p) per chunk position – analogous to the position-wise AdaLN conditioning 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 from an exponentially-decaying distribution $p(d{=}k) \propto e^{-\alpha_d k}$ with α_d=1 over $\{0, \ldots, d_{\max}\}$ and $d_{\max}$=5 (following train-time RTC (Black et al., 2025b); smaller delays more likely), build $\boldsymbol{\tau}^{\star,d}$, fill the front-$d$ slots with ground-truth actions, and mask them out of the loss via $m_p = \mathbf{1}[\tau_p < 1]$, applied after the jitter (Sec. 3.3). Because these

policies condition on n_{obs}=2 observations, the chunk's first position is an action already executed at the previous observation; being history, it is clamped clean alongside the d in-flight actions, so the clean prefix is $d+n_{\text{obs}}-1$ and the ramp width $H-2d-(n_{\text{obs}}-1)$. 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).

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.

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. 1.

Table 1: **Simulation training and inference hyperparameters** for the Leap Cube Reorientation Conditional U-Net 1D policies. The first block holds settings common to every variant; the second lists axes all variants share, with $\pi\mathbf{R}^2$'s value and the baselines' deviations; the last two hold parameters that exist for one method only.

Parameter	$\pi\mathbf{R}^2$	Baseline deviations
Shared across all variants		
Chunk length H	16	
Observation history n_{obs}	2	
Control rate	20 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)	1024	
GPUs	1× A5000	
Training budget (epochs)	800 (zero-delay), 100 (async)	
Shared axes, per-method values		
Train-time $d_{\max}$	5	10 (RTC, FASTER); n/a (Flow, SDP)
Delay distribution $p(d)$	$\propto e^{-\alpha_d d}$, α_d=1	same (RTC); $\text{Uniform}[0, d_{\max})$ (FASTER); n/a (Flow, SDP)
Warm-up / mixing prob.	0.2	0.5 (FASTER, HAS vs. constant-τ); n/a (Flow, RTC, SDP)
Inference budget per call (NFE)	1	15 (Flow, RTC, w/o staircase); $2/7/9$ at d_0=$1/2/3$ (FASTER); $15/7/5/3$ per *action* at T_a=$1/2/3/5$ (SDP)
Deployment	streaming + inpaint	stall (SDP), streaming + inpaint (others)
FASTER only		
Schedule shape α_F	0.6	
Schedule offset u_0	0.9	
SDP only		
Emit width T_a	$\{1, 2, 3, 5\}$: one model trained per delay	
Staircase resolution N	100 diffusion steps	

Baselines. All five variants share the dataset, optimizer, and training budget. Each cell reports the best checkpoint per seed. $\pi\mathbf{R}^2$ 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 $\alpha_d{=}1$ (smaller d more likely following (Black et al., 2025b)), and the front-d positions are clamped to ground-truth actions and masked out of the loss. The five variants then differ only in:

- **Flow**: no d sampling and no front clamp; a single shared noise level $\tau \sim \mathrm{Uniform}[0, 1]$ is applied to the entire chunk.
- **Train-time RTC** (Black et al., 2025b): same per-batch d sampling as $\pi\mathbf{R}^2$, but the noise schedule applies a single shared noise level $\tau \sim \mathrm{Uniform}[0, 1]$ 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 $\alpha{=}0.2$ standard-flow warm-up branch (with probability 0.2 the batch instead uses a single $\tau \sim \mathrm{Uniform}[0, 1]$ 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's FiLM modulation is conditioned position-wise, yielding $\{(\gamma_p, \beta_p)\}_{p=0}^{H-1}$ from a shared projection.
- **FASTER** (Lu et al., 2026): the horizon-aware schedule assigns position p its own flow time $u_p = (1 - j_p^{\alpha_F})\, u_0$ with $\alpha_F{=}0.6$, $u_0{=}0.9$, so later positions converge later. Delay is drawn $d \sim \mathrm{Uniform}[0, d_{\max})$ with $d_{\max}{=}10$, and with probability 0.5 the batch instead uses a single shared τ (their HAS-vs-constant mixing). Deployed through the same clamp-and-commit pipeline as RTC; only the per-position $\mathrm{d}t$ differs. At test time the $K{=}15$ schedule is *truncated* to the step at which the emitted slice has converged, giving $2/7/9$ NFE at $d_0{=}1/2/3$.
- **SDP** (Høeg et al., 2024): a progressive-denoising buffer in which position p is held at a fixed noise level and advanced one step per call. The emit width T_a is baked into the checkpoint (it sets the staircase granularity $h = \lfloor (H - (n_{\mathrm{obs}} - 1))/T_a \rfloor$ over $N{=}100$ diffusion steps), so we train one model per delay, $T_a = d_0 \in \{1, 2, 3, 5\}$. SDP denoises only the front $(n_{\mathrm{obs}} - 1) + T_a$ buffer positions, so an async lookahead slice would emit unconverged actions; we therefore deploy it with *stall*: the robot holds (zero delta) while the chunk is in flight.

Ablations. The three ablations of Fig. 3 start from the $\pi\mathbf{R}^2$ variant above and change one thing each. Their effective delays follow from what each one can and cannot decouple:

- **w/o async slow channel**: identical training, but the slow features are refreshed synchronously with the action head at deployment, so the per-call delay is the whole pipeline with one denoising step, $d = \lceil 1.0\, d_0 \rceil$, with no visual staleness ($d_{\mathrm{vis}} = 0$).
- **w/o staircase**: the asynchronous split is kept ($d = \lceil 1.0\, d_0 \rceil$, $d_{\mathrm{vis}} = \lceil 0.75\, d_0 \rceil$), but the ramped interior is replaced by train-time RTC's single shared noise level, so several denoising steps are still needed per call.
- **w/o latency-adaptive**: trained with $d_{\max}{=}0$, i.e. the schedule is fixed and cannot be parameterized by the delay. Deployed in its native zero-latency mode ($d = \lceil 0.25\, d_0 \rceil$, $d_{\mathrm{vis}} = \lceil 0.75\, d_0 \rceil$) it must stall while a chunk is in flight, rather than conditioning on the in-flight actions.

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. 8). We fine-tune on $8\times$ NVIDIA RTX A5000/A6000 (24/48 GB VRAM). At deployment, the baselines run on a single RTX A5000. $\pi\mathbf{R}^2$ places the slow VLM worker and the fast DiT action head on separate RTX A5000s, emulating a server and edge setup. Note that the baselines' latency is a serial chain (observation $\rightarrow$ backbone $\rightarrow$ denoising $\rightarrow$ action), so extra devices can't shorten the latency of any executed action.

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

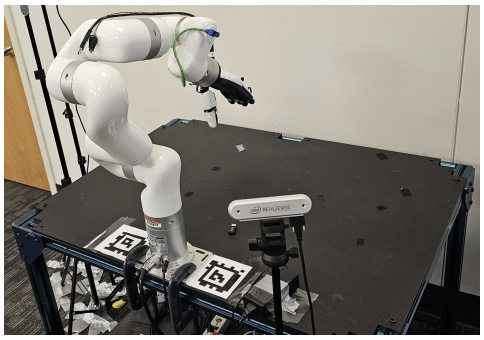


Figure 8: **Real-world workspace.** xArm6 + XHand with a single overhead RGB camera.

Base model. We fine-tune NVIDIA GR00T-N1.7 (Bjorck et al., 2025). 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. 3) 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. 2).

Table 2: **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.

Table 3: **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`

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 $d_{\text{vis}} \sim \text{Uniform}\{0, \ldots, d_{\text{vis}}^{\max}\}$ 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. 4. 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 DiT's AdaLN modulation is evaluated at each chunk position's own noise level τ_p, so the modulation (γ_p, β_p) varies across positions; this reuses the pretrained projection rather than introducing per-position parameters, so each head starts from a position-uniform schedule and gradually specializes during fine-tuning. All four methods in Fig. 5 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. In the real-world $\pi\mathbf{R}^2$ samples its per-batch delay uniformly, $d \sim \text{Uniform}\{1, \ldots, d_{\max}\}$ with $d_{\max}$=5. The train-time RTC baseline samples $d \sim \text{Uniform}\{0, \ldots, 10\}$ (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. 4.

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 methods of Fig. 5 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. (2023).
- *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.

A.3 Visual Reactivity Under Asynchronous Processing

We further compare the average staleness of slow channel between $\pi\mathbf{R}^2$and baselines. Let d_{vlm} be the backbone forward pass, d_{dn} one denoising step, and K the number of denoising steps per call (Sec. 3.2).

Baseline without slow/fast split. Each call costs $d_{\text{base}} = d_{\text{vlm}} + K\, d_{\text{dn}}$, and the execution horizon must be at least d_{base} or the robot stalls waiting for the next chunk. The observation behind a chunk

Table 4: **Real-world training and inference hyperparameters** for the GR00T-N1.7 fine-tunes. The first block holds settings common to every variant; the second lists axes all variants share, with $\pi\mathbf{R}^2$'s value and the baselines' deviations; the last two hold parameters that exist for one method only.

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 epochs (*Tidy Up Book*, *Insert Box*, *Catch Book*) / 200 epochs (*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)

Table 5: **GR00T-N1.7 per-component inference latency.** Numbers in milliseconds.

Component	RTX A5000
Data processing	13.8
Backbone (Qwen-VL)	46.1
Action head (DiT, 4 NFE)	80.7
per NFE	20.2
Full pipeline (E2E)	140.6 (≈ 7.1 Hz)

is already d_{base} old when the chunk is released, and an action drawn uniformly from the horizon executes a further $0.5\,d_{\text{base}}$ later on average. Executed actions therefore act on vision aged $1.5\,d_{\text{base}}$ on average.

$\pi\mathbf{R}^2$. The per-call delay is a single denoising step, d_{dn}, so the minimum execution horizon is also d_{dn}, contributing $1.5\,d_{\text{dn}}$ by the same argument. The slow channel runs continuously and independently: a cached feature computed from an observation at time t becomes available at $t + d_{\text{vlm}}$ and is replaced at $t + 2\,d_{\text{vlm}}$, so its age when used is uniform on $[d_{\text{vlm}}, 2\,d_{\text{vlm}}]$, i.e. $1.5\,d_{\text{vlm}}$ on average. Executed actions therefore act on vision aged $1.5(d_{\text{vlm}} + d_{\text{dn}})$.

Comparison. Since $d_{\text{dn}} < K\,d_{\text{dn}}$ for $K > 1$, $\pi\mathbf{R}^2$'s $1.5(d_{\text{vlm}} + d_{\text{dn}})$ is strictly smaller than the baseline's $1.5(d_{\text{vlm}} + K\,d_{\text{dn}})$. With the measured GR00T-N1.7 costs (Tab. 5: $d_{\text{vlm}} \approx 60$ ms, $d_{\text{dn}} \approx 20$ ms, $K{=}4$) that is 120 ms against 210 ms: the asynchronous split leaves executed actions acting on vision roughly 90 ms *fresher*, not staler. The gain comes from removing $K{-}1$ denoising steps from the critical path, not from refreshing vision faster.

A.4 Training and Inference Algorithms

Adapting to a change in d. The buffer is never rebuilt when the measured delay changes; only the *targets* move. At each call we recompute $\boldsymbol{\tau}^{\star,d}$ from the current d (Eq. 3), then set each position's

end-of-call target to

$$\tilde{\tau}_p = \begin{cases} 1 & p < d \quad \text{(slides out this call)} \\ \tau^{\star,d}_{p-d} & p \geq d \end{cases} \tag{5}$$

and advance it by the clamped deficit $\Delta_p = \max(0,\ \tilde{\tau}_p - \tau_p)$, split over the K Euler substeps of the call: $\tau_p \leftarrow \tau_p + \Delta_p/K$ with the corresponding update to $\mathbf{x}_p$. Positions already at or above their target have $\Delta_p = 0$ and are frozen. A larger d therefore asks more of the interior positions in that call and a smaller d asks less; no state is discarded or reinitialised, so the buffer stays on-manifold across a change in d. On the robot d is the rolling-mean estimate floored at the chunk length (Alg. 2), so an underestimate is corrected on the following call, and if a chunk has not landed when the controller needs an action the last freshly denoised action is re-issued for one control tick rather than reading into the noise tail.

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 $(\mathbf{o}^{\text{fast}}_t, \mathbf{o}^{\text{slow}}_{t:t-T_{\text{slow}}+1})$, model v_θ
Require: Hyperparams $d_{\max}, d^{\max}_{\text{vis}}, j, \alpha$

$u \sim \text{Uniform}[0,1]$
if $u < \alpha$ **then** ▷ warm-up branch: standard flow
  $t \sim \text{Uniform}[0,1]$; $\tau_p \leftarrow t,\ \forall p$ ▷ simulation; real-world draws $t \sim \text{Beta}(1.5, 1)$
  $\mathbf{x}_{\tau,p} \leftarrow (1-\tau_p)\boldsymbol{\epsilon}_p + \tau_p \mathbf{a}_p$
  $m_p \leftarrow 1,\ \forall p$ ▷ all positions contribute to loss
else ▷ staircase branch
  $d \sim p(d{=}k) \propto e^{-\alpha_d k}$ on $\{0, \ldots, d_{\max}\}$ ▷ simulation; real-world uses $\text{Uniform}\{1, \ldots, d_{\max}\}$
  Build $\boldsymbol{\tau}^{\star,d}$ as in Sec. 3.3 ▷ three-region staircase
  $\delta \sim \text{Uniform}[-j, j]$ with $j = \frac{1}{4(H-2d)}$; $\tau_p \leftarrow \text{clip}(\tau^{\star,d}_p + \delta, 0, 1)$ ▷ one draw per example, shared across positions
  $\mathbf{x}_{\tau,p} \leftarrow (1-\tau_p)\boldsymbol{\epsilon}_p + \tau_p \mathbf{a}_p$
  $m_p \leftarrow \mathbf{1}[\tau_p < 1]$ ▷ applied after jitter: a front slot jittered off $\tau{=}1$ re-enters the loss
end if
$d_{\text{vis}} \sim \text{Uniform}\{0, \ldots, d^{\max}_{\text{vis}}\}$ ▷ slow-channel staleness (real-world only)
$\mathbf{o}^{\text{slow}} \leftarrow \mathbf{o}^{\text{slow}}_{t-d_{\text{vis}}}$; $\mathbf{z}^{\text{act}} \mathrel{+}= e(d_{\text{vis}})$ ▷ added to action tokens, broadcast over H
$\hat{\mathbf{v}}_p \leftarrow v_\theta(\mathbf{x}_\tau, \boldsymbol{\tau}, \mathbf{o}^{\text{fast}}_t, \mathbf{o}^{\text{slow}})_p$
$\mathcal{L} \leftarrow \sum_{p=0}^{H-1} m_p \left\| \hat{\mathbf{v}}_p - (\mathbf{a}_p - \boldsymbol{\epsilon}_p) \right\|^2$
Update θ on $\mathcal{L}$

A.5 Additional Reactivity Analysis

The main-text reactivity comparison (Fig. 6) is on *Tidy Up Book*; the same pattern holds across the other tasks (Fig. 9). 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.

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
$s_{\text{last}} \leftarrow 0$ ▷ index at the last swap; $s_{\text{last}} + d$ is the next swap target
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 $i \geq s_{\text{last}} + d$ **then** ▷ the d freshly denoised actions $[d, 2d)$ are spent
    if new chunk $\mathbf{C}_{\text{new}}$ is available from `Action_Worker` **then**
      $\mathbf{C} \leftarrow \mathbf{C}_{\text{new}};\ i \leftarrow d;\ s_{\text{last}} \leftarrow i$ ▷ swap and skip past in-flight front actions
    else
      $i \leftarrow \max(0,\ i - 1)$ ▷ not landed: hold the last freshly denoised action, never the noise tail
    end if
  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
    $\mathbf{C}_{\text{new}} \leftarrow v_\theta.\texttt{euler_step}(\mathbf{a}^{\text{inflight}}, \mathbf{o}^{\text{fast}}, \mathbf{o}_{\text{cached}}^{\text{slow}}, e(d_{\text{vis}}))$ ▷ 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)
    Automatically update cache with $(\mathbf{o}_{\text{cached}}^{\text{slow}}, t_{\text{image}})$
  end loop
end procedure

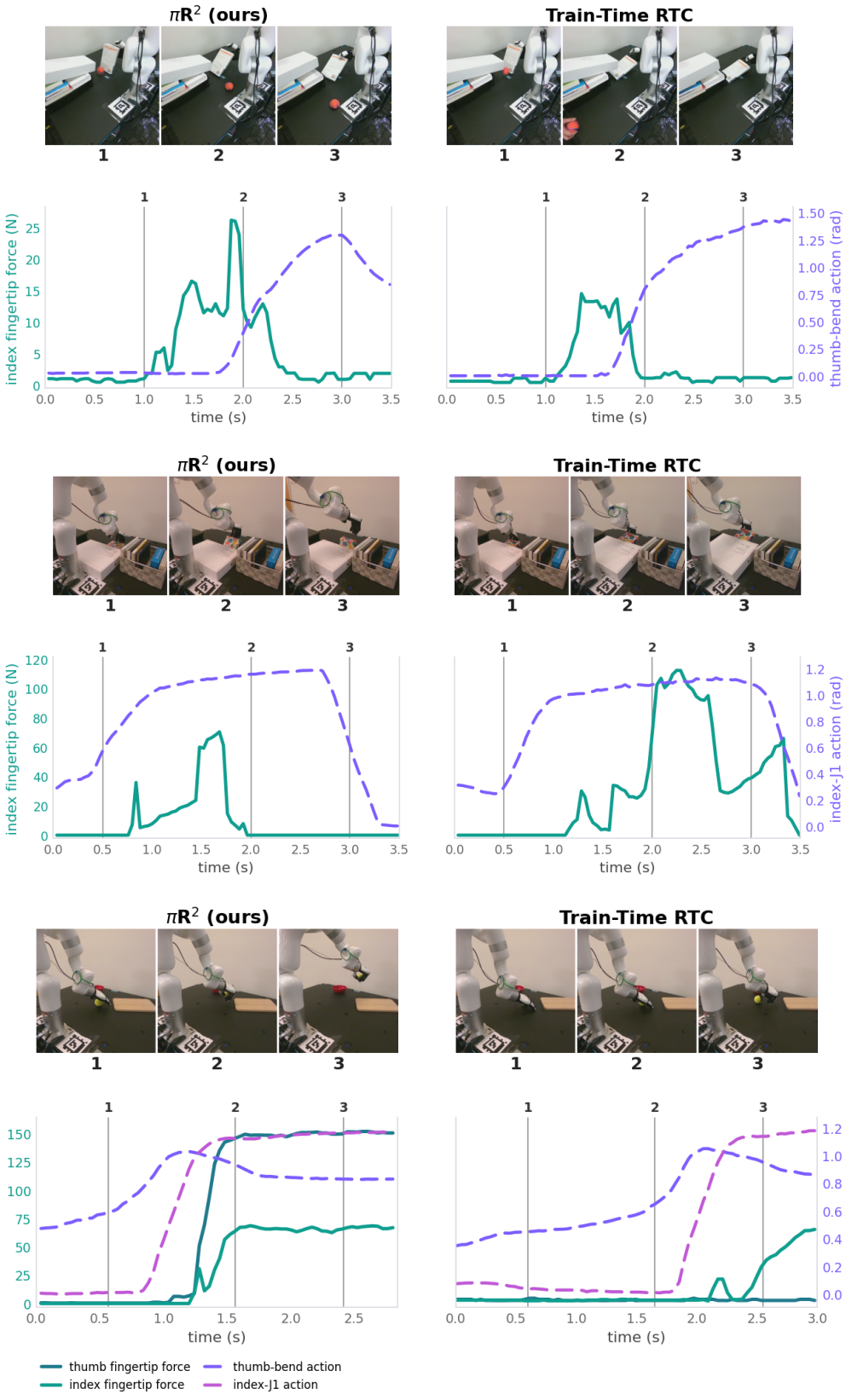


Figure 9: **Reactivity on the remaining tasks.** Fingertip force (solid, left axis) and the emitted action (dashed, right axis) over time for $\pi\mathbf{R}^2$ and Train-Time RTC, with overhead frames at the marked times. Top: *Catch Book*. Middle: *Insert Box*. Bottom: *Don't Spill*.