DUAL-EXPERT FLOW MATCHING FOR DYNAMIC ROBOT MANIPULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models predict where a robot should move, but not how fast. The velocity profile is neither supervised nor predicted; the low-level controller reconstructs it from position targets. For quasi-static manipulation this is harmless. For dynamic manipulation, in which the robot tosses, flips, slides, or pushes, the velocity at contact decides the outcome, and position tracking distorts it. We show that accurate position does not imply accurate velocity and we establish the need to learn velocity from the demonstrations. We propose VIPER, which adds a velocity expert to a pretrained VLA and denoises velocity jointly with the VLA's position expert from shared noise to ensure temporal alignment between the noise-free position-velocity actions. VIPER leaves the VLA's backbone frozen and attaches to any VLA with a flow-matching action expert. We evaluate VIPER on three VLA backbones and six dynamic manipulation tasks in simulation and on a real robot, from tossing and flipping to catching and handover. VIPER raises mean success from 32% to 51%, and improves backbones that already reason about temporal context. Our project website is available at https://viper-robot.github.io/.

1 INTRODUCTION

Dynamic manipulation is manipulation in which the robot's velocity, not only its position, decides the outcome. A robot can push a ball in exactly the right direction and still miss the basket. The robot's velocity at the moment of contact sets the ball's flight (Fig. 1, left). Tossing a bottle, flipping bread in a skillet, and catching a dropped ball share the same structure. The robot must reach the right configuration, and the velocity at which the robot arrives decides success. Vision-Language-Action (VLA) models are now taking on dynamic manipulation, and the VLA action representation leaves one question unanswered. **How fast should the robot move?**

A *position-only* VLA does not predict velocity at all; the low-level controller chooses the velocity, and chooses wrong for dynamic tasks. The VLA outputs a chunk of *joint-position targets* at a fixed action rate (Black et al., 2024; PI et al., 2025), and the controller moves the robot from one target to the next. The standard controller tracks each target with a desired velocity of zero. For a *quasi-static* task zero desired velocity is the right choice, because the robot should hold still between chunks. For a dynamic task zero desired velocity is the wrong choice, because the robot should build momentum across several targets. The controller instead accelerates toward and decelerates around every target. We show the result in Fig. 1A: the position tracks the demonstration while the velocity oscillates around it.

Differentiating the predicted positions does not recover the demonstrated velocity, and sampling positions more densely does not help. Fitting a spline through the position targets and differentiating the spline gives the controller a *velocity feedforward* term and removes the oscillation (Hechtl et al., 2026). Nevertheless, such differentiation can amplify the errors by a factor that grows with the sampling rate. Our studies show that a closer position fit does not yield a better velocity. We show the consequence on the ball push in Fig. 1B: the spline matches the position targets closely and still underestimates the velocity peak that decides the outcome. This motivates the need to learn velocity from the demonstrations rather than recover velocity from predicted positions.

We propose VIPER, which adds a *velocity expert* to a pretrained VLA and generates velocity together with the VLA's *position expert*, so that the two stay temporally aligned. We train the

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.

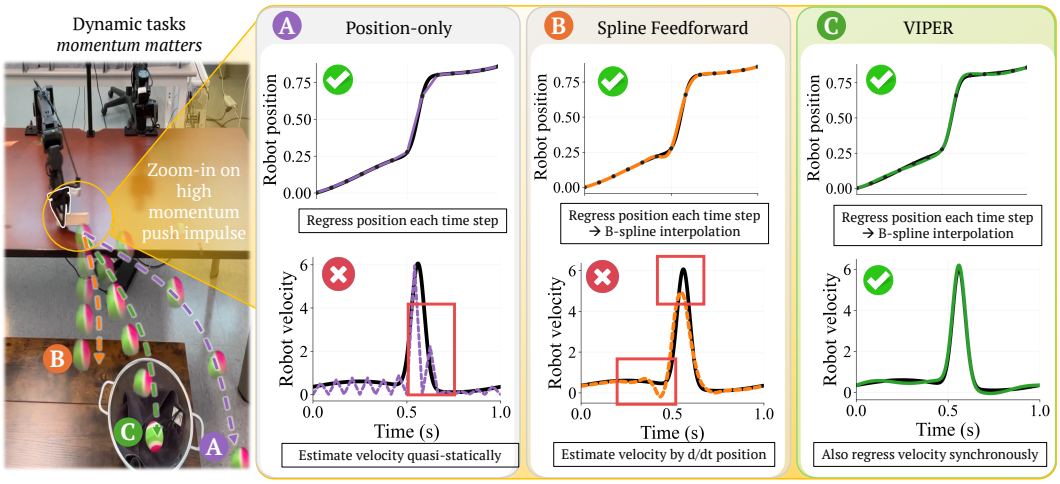

Figure 1: **Velocity at contact decides the outcome, and only a policy that predicts velocity reproduces it.** All three methods track the demonstrated position (top row, black) and only C tracks the demonstrated velocity (bottom row, black). Compare the velocity peaks: (A) position-only execution oscillates around the peak, (B) a spline differentiated through the position targets undershoots it, and (C) VIPER matches its magnitude and timing. The ball's path in the photo follows the velocity, not the position: A overshoots the basket, B falls short, C lands. Dots are the position targets the VLA predicts; colored curves are the executed motion.

velocity expert on the joint velocities recorded in the demonstrations, alongside the position expert and on the same frozen backbone, so VIPER attaches to any VLA with a flow-matching action expert. A separate expert is a necessity rather than a convenience: our ablations show that a single expert trained to predict position and velocity together reproduces neither.

Predicting velocity introduces a second requirement: position and velocity must describe the same stage of motion. Two independently generated trajectories can each be plausible while giving the controller conflicting references. In the ball push, a velocity peak of the correct magnitude that arrives before the robot reaches the ball cannot deliver the push. We therefore couple the two experts so that position and velocity are denoised together along a shared path, in training and at inference. Sharing the noise has a concrete effect that is visible in the interpolants: subtracting the two paths cancels the noise exactly, so the joint state carries the noise-free position-velocity offset at every point along the path, and under a first-order phase model that offset is what identifies the *temporal phase*. We show the result on the ball push in Fig. 1C: the velocity peak has the right magnitude and arrives at the right time.

VIPER improves every VLA backbone we test, in nearly all experiments, on dynamic manipulation tasks in simulation and on a real robot. We evaluate $\pi_{0.5}$, RTC, and TEMPO on four non-prehensile, momentum-driven tasks: tossing a bottle, flipping bread in a skillet, sliding a cube into a second cube, and pushing a ball into a basket. Two further tasks require timing against a moving object: catching a dropped ball and taking a bottle from a walking person. VIPER raises mean success from 32% to 51% across the eighteen backbone–task combinations. Spline velocity feedforward recovers only part of these gains. The gains hold on TEMPO, which already conditions on temporal context, so predicting velocity is complementary to perceiving motion. Ablations confirm that both the separate velocity expert and the coupling are needed.

In summary, we make four contributions. (1) We show that dynamic manipulation needs velocity, and that a position-only VLA loses it. (2) We give an exact account of what differentiating a fitted position trajectory does to prediction error, showing that position accuracy does not control velocity accuracy and that denser sampling does not help, and we confirm this empirically for the interpolating fit used in prior work. (3) We propose VIPER, a velocity expert added to a pretrained VLA and coupled to its position expert through shared noise, with an analysis of why coupling supplies the information needed for temporal alignment and an empirical measurement of the alignment it produces. (4) We show that VIPER delivers large gains across three VLA backbones and six dynamic manipulation tasks.

2 Related Work

Vision-Language-Action Models. VLAs pair a pretrained vision-language backbone with an action head and fine-tune the pair on robot demonstrations (Brohan et al., 2022; 2023; Kim et al., 2024; Black et al., 2024; PI et al., 2025; Physical Intelligence et al., 2026; O'Neill et al., 2024; Kim et al., 2025; Pertsch et al., 2025). Early VLAs emit actions as discrete tokens through the language model, and recent VLAs attach a continuous action head (Team et al., 2024; Li et al., 2024; Liu et al., 2025; Wen et al., 2025; Shukor et al., 2025; Bjorck et al., 2025; Zheng et al., 2026) that predicts a chunk of position targets at each inference step; in both cases the action is a position. Unlike work that improves the backbone or its training data, we keep the backbone and the demonstrations fixed and ask whether the action head should also predict velocity.

Dynamic Manipulation with VLAs. Recent work extends VLAs to dynamic manipulation (Xie et al., 2026; Syed et al., 2026; Sendai et al., 2025; Shi et al., 2026; Lu et al., 2026; Guo et al., 2026) by changing when the policy acts and what the policy sees. Real-Time Chunking executes flow-based action chunks asynchronously so that inference latency no longer stalls the robot (Black et al., 2026), and VLASH streams actions during continuous inference to keep up with moving objects (Tang et al., 2025). TEMPO conditions the policy on visual motion features and proprioceptive history to resolve the state aliasing of single-frame observations (Feng et al., 2026). Complementary to these improvements in temporal perception and execution timing, which leave the action representation unchanged, VIPER changes what the policy outputs by adding velocity, and our experiments show that the two kinds of improvement can compound.

Flow Matching for Robot Actions. Generative action heads model the distribution over action chunks rather than a single regression target. Diffusion Policy denoises action sequences with a conditional diffusion model (Chi et al., 2025), and flow matching learns a vector field that transports noise to the action distribution (Lipman et al., 2022). For VLAs, the π family attaches a flow-matching action expert to the backbone and generates position chunks conditioned on vision and language, and this action expert is the one VIPER builds on (Black et al., 2024; PI et al., 2025). Complementary to a single flow over position, VIPER supervises velocity with the joint velocities recorded in the demonstrations and generates position and velocity with two experts whose flows are coupled. Sec. 4 describes the coupling.

Velocity in Trajectory Generation and Control. Classical trajectory optimization produces time-parameterized trajectories, so velocity and acceleration are defined along the motion by construction (Ratliff et al., 2009; Schulman et al., 2013). Learning-based policies instead emit position targets at a fixed action rate and leave the motion between targets to interpolation and tracking, typically with zero desired velocity (Luo et al., 2024; Pro et al., 2026). For VLAs, one line of work fits a cubic B-spline to the predicted positions and differentiates the spline to obtain a velocity feedforward term (Hechtl et al., 2026). Another predicts B-spline control points directly, which reduces jitter and allows faster execution (Yang et al., 2026; Han et al., 2026). FlowMP takes the same construction into motion planning: its velocity and acceleration targets are derivatives of B-splines fitted to optimized via-points, over which it then learns a generative prior (Nguyen et al., 2025). In all of these methods the executed velocity is derived from the predicted positions, and we compare against the most recent VLA form of this idea, spline velocity feedforward (Hechtl et al., 2026), in Sec. 5. In contrast, VIPER treats the demonstrated joint velocity as a supervised output of the policy, and Sec. 3 shows why the derived velocity can miss what the demonstration contains.

3 Can Velocity Be Recovered from Predicted Positions?

Figure 1 illustrates why velocity is critical for dynamic manipulation. Here, we analyze why velocity recovered from predicted positions can differ from the demonstrated motion. We consider two common strategies: quasi-static position-only execution (Sec. 3.1) and velocity feedforward from a differentiated position trajectory (Sec. 3.2). The former suppresses task-critical velocity structure, while the latter amplifies high-frequency error; neither reliably recovers the demonstrated velocity from positions alone.

Let o denote the current observation and l the language instruction. Let $\mathcal{D}$ denote the demonstration dataset. A demonstration provides a future action chunk $\mathbf{a} = (\mathbf{Q}, \dot{\mathbf{Q}}) \sim \mathcal{D}$, where $\mathbf{Q} = (q_1, \ldots, q_H)$

and $\dot{\mathbf{Q}} = (\dot{q}_1, \ldots, \dot{q}_H)$ denote the demonstrated joint-position and joint-velocity trajectories, sampled at rate f_s. A position-only VLA predicts $\hat{\mathbf{Q}} = f_\theta(o, l)$ and is optimized for position accuracy.

3.1 Quasi-Static Position Execution

A position-only policy does not model how fast the robot should move. Existing impedance-control implementations therefore commonly track position with zero desired velocity (Hechtl et al., 2026; Pro et al., 2026; Luo et al., 2024). This is a natural choice for quasi-static manipulation, but not for dynamic tasks that require sustained or precisely timed motion.

To see why, consider a single joint tracking the k-th position target q_k over one action interval $[t_k, t_{k+1})$ of length $\Delta t = 1/f_a$, where f_a is the action rate. With desired velocity set to zero, and a PD controller with gains K_p and K_d, the tracking dynamics are:

$$m\,\ddot{q} + K_d\,\dot{q} + K_p\,(q - q_k) = 0, \qquad t \in [t_k, t_{k+1}), \tag{1}$$

whose unique equilibrium is $(q, \dot{q}) = (q_k, 0)$. ***Rest at the target is an attractor of every interval.*** Writing $\Delta q_k = q_k - q_{k-1}$ and taking the critically damped case with natural frequency $\omega = \sqrt{K_p/m}$, the executed velocity within the interval is:

$$\dot{q}(s) = \Delta q_k\,\omega^2 s\,e^{-\omega s}, \qquad s = t - t_k, \qquad \dot{q}_{\max} = e^{-1}\,\omega\,\Delta q_k, \tag{2}$$

which vanishes at $s = 0$ and decays back to zero. Two consequences follow. First, the executed velocity is driven toward zero once per action interval, so the velocity profile carries ripple at the action rate f_a rather than the temporal structure of $\dot{\mathbf{Q}}$. Second, comparing equation 2 with the mean velocity $\Delta q_k/\Delta t$ the interval must realize, the ripple magnitude is governed by the dimensionless ratio $\frac{\dot{q}_{\max}}{\Delta q_k/\Delta t} = e^{-1}\,\omega\,\Delta t$. That is, even in the most favorable case, where the controller realizes the correct mean velocity on every interval, the best achievable velocity is the interval-averaged demonstration, i.e. a moving average of width Δt applied to $\dot{\mathbf{Q}}$. This is especially important when the task requires the robot to build or maintain momentum over several consecutive waypoints, such as during a push, toss, or release: the position targets may be correct while the velocity at contact is set by ω, Δt, and Δq_k rather than by the demonstration (Fig. 1A).

3.2 Velocity from Position Differentiation

An alternative, explored for example in Hechtl et al. (2026), is to reconstruct a continuous trajectory using a cubic B-spline from the predicted positions and differentiate it to obtain a velocity reference.

For analysis, we denote the ground-truth position profile underlying $\mathbf{Q}$ by $S_h(t)$, which we represent as a degree-p B-spline with uniform knot spacing h, and let $\widehat{S}_h$ be the spline fitted to $\hat{\mathbf{Q}}$ on the same knot grid:

$$S_h(t) = \sum_i c_i\,B_p\big(\tfrac{t-t_i}{h}\big), \qquad \widehat{S}_h(t) = \sum_i \hat{c}_i\,B_p\big(\tfrac{t-t_i}{h}\big), \tag{3}$$

where B_p is the cardinal B-spline of degree p; $S_h(t)$'s derivative S_h' recovers the demonstrated velocity profile $\dot{\mathbf{Q}}$. Analogously, the position-derived velocity reference is obtained analytically as $\widehat{S}_h'(t) = \frac{d}{dt}\widehat{S}_h(t)$, where the velocity is determined by the reconstructed position trajectory $\widehat{S}_h(t)$.

To see what differentiation can and cannot recover, write the coefficient error as $e_i = \hat{c}_i - c_i$, so the position and velocity errors are

$$\widehat{S}_h - S_h = \sum_i e_i B_p\big(\tfrac{t-t_i}{h}\big), \qquad \widehat{S}_h' - S_h' = \frac{1}{h}\sum_i (e_i - e_{i-1})\,B_{p-1}\Big(\tfrac{t-t_{i-1/2}}{h}\Big), \tag{4}$$

where the second expression uses $B_p' = B_{p-1}(\cdot + \tfrac{1}{2}) - B_{p-1}(\cdot - \tfrac{1}{2})$. Differentiation therefore acts on the prediction error as a ***first difference scaled by*** $1/h$. This has two consequences. First, position accuracy does not control velocity accuracy: the position error depends on e while the velocity error on the differences of e. Second, the amplification is governed by the ***frequency content*** of the error rather than by the sampling rate alone. Refining the grid shrinks h but does not move the residual to lower frequency, which is why a closer position fit need not give a better velocity.

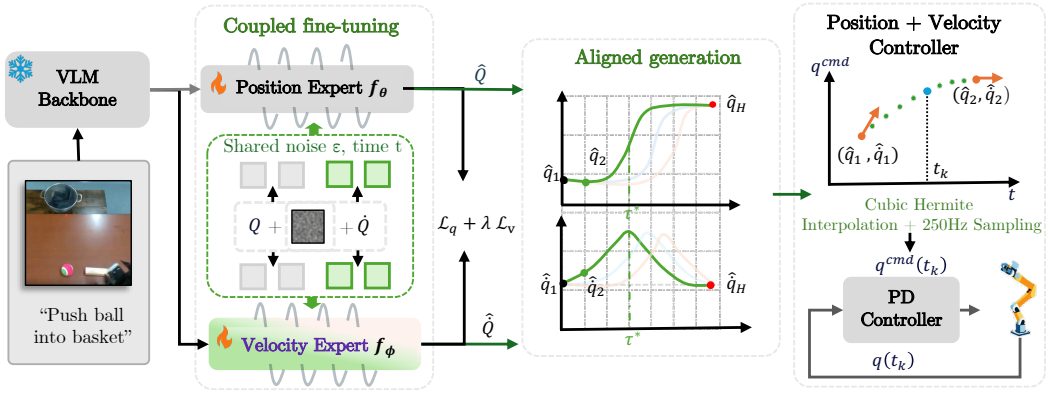

Figure 2: **VIPER Overview.** VIPER adds a velocity expert alongside the position expert and jointly fine-tunes the two using paired position–velocity demonstrations and shared flow noise. The coupled experts generate temporally aligned position and velocity waypoints during inference, which are interpolated with a cubic Hermite spline and tracked by a PD controller.

Empirical observation. We fit cubic B-splines ($p = 3$) to demonstrations sampled at 10–200 Hz and evaluate against the original 250 Hz trajectory (Appendix Table 5); the velocity error remains at 0.8–1.8 rad/s as the sampling rate increases, even though the position error stays small (≈ 0.03) throughout, as equation 4 describes. This motivates us to predict the velocity explicitly rather than recover it from the positions.

4 VIPER: Velocity-Informed Policy Enhancement

Section 3 motivates learning velocity directly from demonstrations. We propose VIPER, a Velocity-Informed Policy Enhancement framework that augments a pretrained VLA with an explicit velocity expert and couples position and velocity generation during training and inference.

4.1 Velocity Expert and Coupled Training

VIPER augments the position expert f_θ with a velocity expert f_ϕ of the same architecture. Both share the VLM conditioning c, while f_θ models joint positions and f_ϕ is supervised directly using the joint velocities recorded in the demonstrations.

Independent generation, however, does not guarantee that the predicted position and velocity describe the same stage of motion. We therefore couple the two flow-matching processes. For each demonstration pair $(\mathbf{Q}, \dot{\mathbf{Q}})$, we sample $t \sim p_{\mathrm{FM}}(t)$ and $\epsilon \sim \mathcal{N}(0, I)$ and construct

$$\mathbf{x}_t^p = (1-t)\mathbf{Q} + t\epsilon, \qquad \mathbf{x}_t^v = (1-t)\dot{\mathbf{Q}} + t\epsilon. \tag{5}$$

The same t and ϵ are used for both trajectories, and both experts condition on the joint state $X_t = (\mathbf{x}_t^p, \mathbf{x}_t^v, t, c)$.

Training proceeds in two stages. We first train the original VLA for position prediction, then freeze the VLM backbone and jointly fine-tune the position and velocity experts with

$$\mathcal{L} = \mathbb{E}\Big[\|f_\theta(X_t) - (\epsilon - \mathbf{Q})\|_2^2 + \lambda_v \|f_\phi(X_t) - (\epsilon - \dot{\mathbf{Q}})\|_2^2\Big]. \tag{6}$$

Why coupling promotes temporal alignment. Because the two interpolants share the same noise, their difference cancels the noise exactly: $\frac{\mathbf{x}_t^v - \mathbf{x}_t^p}{1-t} = \dot{\mathbf{Q}} - \mathbf{Q}$, where $t < 1$.

Thus the joint state exposes a noise-free position–velocity offset that is lost under independent noise. Under the first-order phase model in Appendix C, this offset identifies temporal phase and motivates coupled denoising. This is a design rationale rather than a guarantee; Fig. 5 and Table 4 verify the alignment effect empirically.

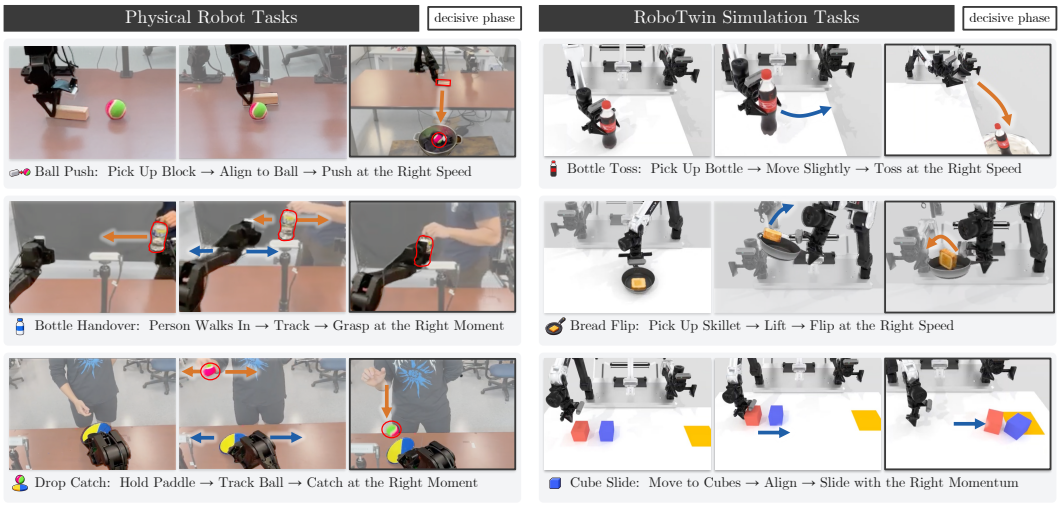

Figure 3: **Dynamic manipulation tasks overview.** We evaluate three physical robot tasks (left, Sec. 5.1) and three RoboTwin simulation tasks (right), each shown as three phases; the outlined frame is the decisive phase, and the line below names it. Blue arrows mark the robot's motion, orange arrows the object's or the person's. See task setup video on our webpage.

4.2 Coupled Inference and Low-Level Controller

At inference, the VLM backbone encodes the current observation and language instruction into the shared conditioning representation c. The position and velocity experts are initialized from the same Gaussian noise $\epsilon_0 \sim \mathcal{N}(0, I)$, $\mathbf{x}_1^p = \epsilon_0, \mathbf{x}_1^v = \epsilon_0$. At each integration step t_m, the two experts are jointly denoised under the same context c and using the shared state $X_{t_m} = (\mathbf{x}_{t_m}^p, \mathbf{x}_{t_m}^v, t_m, c)$.

Both experts are updated using the same integration schedule, yielding the final position and velocity predictions $(\hat{\mathbf{Q}}, \hat{\dot{\mathbf{Q}}})$. The paired trajectories are then post-processed and passed jointly to the low-level controller for execution, where we perform Cubic Hermite interpolation between each consecutive waypoint $(\hat{q}_k, \hat{\dot{q}}_k)$ and $(\hat{q}_{k+1}, \hat{\dot{q}}_{k+1})$. We sample the resulting spline at 250 Hz to obtain commanded positions and velocities $(q^{\mathrm{cmd}}(t_j), \dot{q}^{\mathrm{cmd}}(t_j))$ for a closed-loop PD control.

5 Experiments

We evaluate VIPER on three VLA backbones and six dynamic manipulation tasks, three on a physical robot and three in simulation. Sec. 5.1 describes the tasks, backbones, and metric. Sec. 5.2 reports success rates and the findings the results support. Sec. 5.3 examines where the gain comes from and where VIPER does not help. Sec. 5.4 ablates the two design decisions behind VIPER. Details that do not fit here are in Appendix D.

5.1 Evaluation Setup

Physical robot tasks. We run three tasks on an I2RT YAM arm with 50 trials per method and task (Fig. 3, left). Two are adapted from Tempo-Bench (Feng et al., 2026).
• *Ball Push (velocity at contact):* the robot holds a block and pushes a ball into a basket whose position varies across trials; the ball reaches the basket only if the push velocity is right.
• *Bottle Handover (timing against a moving object):* a person walks past holding a bottle, and the robot must track the bottle and take it at the right moment.
• *Drop Catch (timing against a moving object):* a person moves a ball above a robot-held pad and releases it at an arbitrary time, and the robot must place the pad under the ball before it lands.

RoboTwin simulation tasks. We run three tasks in RoboTwin (Chen et al., 2025) with 100 rollouts per method and task (Fig. 3, right).
• *Bottle Toss (velocity at release):* the robot picks up a bottle and tosses it onto a target; the landing point is set by the velocity at release.

Table 1: **VIPER improves all three backbones on nearly every task.** Success rate in %. Each block is one VLA backbone; compare the three rows within a block. `Position-Only` is the backbone as published, `Spline-FF` adds spline velocity feedforward, and `VIPER` adds the velocity expert. Avg is the mean over the three tasks on each side. The Δ row is `VIPER` minus `Position-Only`. Bold marks the best and italics the second best within a block. 50 trials per physical task, 100 rollouts per simulation task. Video rollout comparisons between methods are available on our webpage.

			Physical Robot Tasks				**RoboTwin Simulation Tasks**			
#	Method	Where the velocity comes from	Ball Push	Bottle Handover	Drop Catch	Avg	Bottle Toss	Bread Flip	Cube Slide	Avg
		$\pi_{0.5}$ backbone *(general-purpose VLA)*								
1	`Position-Only` (PI et al., 2025)	Controller stops at every target	*28*	26	36	30.0	9	20	*20*	16.3
2	`Spline-FF` (Hechtl et al., 2026)	Derivative of a spline through targets	24	*30*	*40*	*31.3*	*40*	*30*	12	*27.3*
3	`VIPER` *(ours)*	Predicted from demonstrations	**52**	**44**	**46**	**47.3**	**57**	**42**	**56**	**51.7**
	Δ Success		+24	+18	+10	+17.3	+48	+22	+36	+35.3
		`RTC` backbone *(asynchronous action-chunk execution)*								
4	`Position-Only` (Black et al., 2026)	Controller stops at every target	*32*	*44*	**50**	*42.0*	9	11	33	17.7
5	`Spline-FF` (Hechtl et al., 2026)	Derivative of a spline through targets	*32*	38	**50**	40.0	*19*	*20*	*45*	*28.0*
6	`VIPER` *(ours)*	Predicted from demonstrations	**36**	**50**	*48*	**44.7**	**55**	**39**	**62**	**52.0**
	Δ Success		+4	+6	−2	+2.7	+46	+28	+29	+34.3
		`TEMPO` backbone *(temporal context for dynamic tasks)*								
7	`Position-Only` (Feng et al., 2026)	Controller stops at every target	**34**	*74*	66	*58.0*	8	25	*42*	25.0
8	`Spline-FF` (Hechtl et al., 2026)	Derivative of a spline through targets	22	*74*	*72*	56.0	*28*	*35*	30	*31.0*
9	`VIPER` *(ours)*	Predicted from demonstrations	*32*	**86**	**86**	**68.0**	**40**	**38**	**56**	**44.7**
	Δ Success		−2	+12	+20	+10.0	+32	+13	+14	+19.7

• *Cube Slide (momentum transfer):* the robot strikes a red cube so that it slides into a blue cube and pushes the blue cube into a target region; both cubes keep moving after the brief contact.
• *Bread Flip (velocity at launch):* the robot lifts a skillet to launch and flip a slice of bread and catch it again; the flip succeeds only within a narrow band of lift velocities.

VLA backbones. We initialize from three VLAs that differ in how they handle dynamic tasks; each is used as published, with its own weights and action expert.
• $\pi_{0.5}$ (PI et al., 2025) (general-purpose VLA): a flow-matching action expert on a pretrained backbone, with no provision for dynamics.
• `RTC` (Black et al., 2026) (asynchronous execution): executes action chunks asynchronously so that inference latency does not stall the robot, and blends each new chunk into the actions already committed.
• `TEMPO` (Feng et al., 2026) (temporal context): conditions the policy on visual motion features and proprioceptive history, and is the strongest of the three on dynamic tasks.

Methods. On each backbone we compare three methods that differ only in where the controller's velocity comes from.
• `Position-Only`: the backbone as published; the controller tracks each predicted position target with zero desired velocity, so the robot stops at every target.
• `Spline-FF` (Hechtl et al., 2026): spline velocity feedforward; a cubic B-spline is fitted through the position targets and its derivative is given to the controller as the velocity.
• `VIPER`: the controller receives the velocity predicted by the velocity expert, trained on the demonstrations.
All three use the same demonstrations, the same backbone weights, and the same low-level controller; the only difference is the velocity the controller receives.

Metric. We report success rate, with success judged by the task-specific goal condition above (Appendix D.1). Camera placement and lighting are fixed across methods within a task. One standard error of a success rate is about 7 points over 50 trials and 5 points over 100 rollouts, so differences of a few points are within evaluation noise.

5.2 Quantitative Results

VIPER improves every backbone, in nearly all experiments (Table 1). `VIPER` has the highest average on both sides of the table for all three backbones, and every average gain is larger than one standard error except `RTC` on the physical tasks (+2.7), which we return to in Sec. 5.3. Across

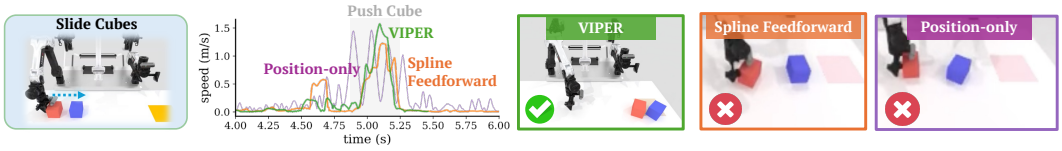

Figure 4: **VIPER reproduces the speed peak at contact; the spline does not.** A Cube Slide rollout on $\pi_{0.5}$. Compare the end-effector speed inside the shaded push window: `Position-Only` oscillates, `Spline-FF` is smooth but peaks too low, and `VIPER` reaches the demonstrated peak. The three end states on the right show that only the full peak sends the blue cube into the target.

the eighteen backbone–task combinations `VIPER` beats `Position-Only` in sixteen and raises mean success from 32% to 51%. The two exceptions, `RTC` on Drop Catch and `TEMPO` on Ball Push, are two points below the baseline, inside evaluation noise.

The largest gains are on tasks where momentum must be generated (Table 1). On Bottle Toss, `VIPER` lifts $\pi_{0.5}$ from 9% $\rightarrow$ 57%, and on the physical Ball Push from 28% $\rightarrow$ 52%. These are the tasks in which the position targets say where to move but nothing about how hard to strike or release; the same trend holds on every backbone, with simulation averages rising by 20 to 35 points.

Spline feedforward recovers only part of the gain (Table 1). Comparing the `Spline-FF` and `VIPER` rows of each block, `VIPER` wins seventeen of eighteen combinations. Cube Slide with $\pi_{0.5}$ is the sharpest case: `Spline-FF` falls below `Position-Only` (20% $\rightarrow$ 12%) while `VIPER` reaches 56%. A smoother velocity is not the same as the demonstrated velocity.

Velocity helps the strongest dynamic-manipulation backbone too (Table 1). `TEMPO` (rows 7–9) is the strongest baseline on the physical tasks, with a 58% average against 42% for `RTC` and 30% for $\pi_{0.5}$, and `VIPER` still raises that average to 68%, with Drop Catch going 66% $\rightarrow$ 86%. Knowing when to act and knowing how fast to move are different pieces of information, and `TEMPO` supplies only the first.

5.3 Analysis

The gain comes from the velocity peak, not from smoothing (Fig. 4). In the Cube Slide rollout shown, all three methods follow the same position targets, and `Spline-FF` removes the oscillation of `Position-Only`. Only `VIPER` reaches the demonstrated speed inside the push window, and only that rollout succeeds. Smoothing the position trajectory and reproducing the demonstrated velocity are different things, and the task rewards the second. See videos on our webpage.

Velocity references also shorten execution relative to `Position-Only` (Appendix D.2). On $\pi_{0.5}$ in simulation, `VIPER` completes successful episodes faster than `Position-Only` on every task (Bottle Toss 5.0 s $\rightarrow$ 4.0 s), because the controller no longer decelerates toward every position target. We report this comparison for the two methods we timed; it does not separate `VIPER` from `Spline-FF`, which also removes the stop-and-go.

Where VIPER helps least: timing tasks on a backbone that already handles timing (Table 1). `RTC` (rows 4–6) gains 34 points on average in simulation but only 3 on the physical robot. The physical tasks are where `RTC`'s asynchronous execution already earns its keep: on the two timing tasks its `Position-Only` baseline is 8 to 18 points above $\pi_{0.5}$, so less of the failure is left for velocity to fix. `RTC` also blends each new action chunk into the actions already committed, which smooths the transition between chunks and, we expect, damps the short velocity peak that VIPER supplies; the simulation tasks, whose peaks fall inside a chunk, do not pay this cost. The two small losses in Table 1 are both on timing tasks and both within one standard error.

5.4 Ablations

We ablate the two design decisions behind VIPER on the three simulation tasks with $\pi_{0.5}$: two experts rather than one, and coupled rather than independent generation (Table 2).

Coupling the experts is what makes two experts work (Table 2, Fig. 5). Coupling lifts the same two-expert architecture on every task, most on Bottle Toss (28% $\rightarrow$ 57%). Fig. 5 shows the failure that coupling removes: on Bread Flip the independent velocity expert places its peak before

Table 2: **Both a separate velocity expert and the coupling are needed.** Success rate in % on the simulation tasks with $\pi_{0.5}$. A single expert that predicts position and velocity together is the weakest; two independent experts recover some of the gap; coupling the two experts (`VIPER`) is best on every task. Bold marks the best and italics the second best.

#	Method	Bottle Toss	Bread Flip	Cube Slide
1	`Joint-Expert` (one head for position and velocity)	*34*	15	20
2	`Independent-Experts` (two heads, no coupling)	28	*38*	*47*
3	`VIPER` (two heads, coupled)	**57**	**42**	**56**

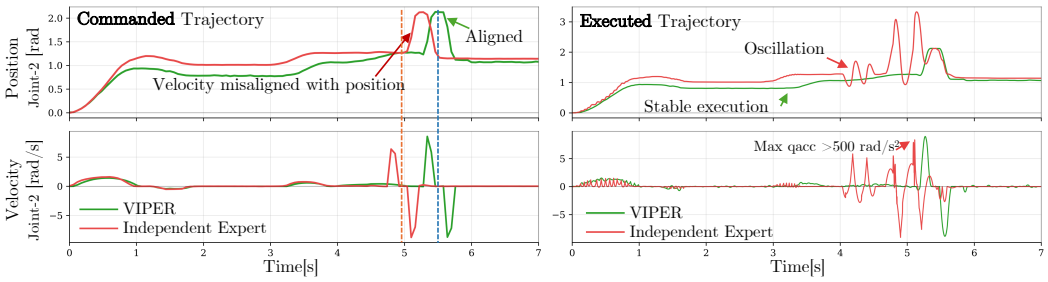

Figure 5: **Independent experts put the velocity peak at the wrong time; coupling aligns it.** Bread Flip, joint 2, commanded (left) and executed (right) trajectories. With `Independent-Experts` the commanded velocity peak arrives before the commanded position moves (dashed lines), and the executed motion oscillates with acceleration spikes above 500 rad/s^2. With `VIPER` the peak coincides with the position change and execution is stable.

the position expert moves, and the controller, given a position and a velocity that describe different moments, oscillates. We quantify this for the two dual-expert variants with a position–velocity consistency error, the gap between the predicted velocity and the velocity implied by the predicted positions (Appendix D.3); coupling reduces it on every task, by 64% on Bottle Toss. **One expert for both signals reproduces neither** (Table 2, Appendix D.4). Appending velocity to the position expert's output and training the two jointly gives the weakest model on every task (Bread Flip 15%, against 42% for `VIPER`). Its predicted position drifts from the demonstration and its predicted velocity is non-zero while the demonstration is stationary, so the robot moves before the interaction it is supposed to prepare for.

Denser position sampling does not let the `Spline-FF` fit recover the demonstrated velocity (Appendix D.5). We give the `Spline-FF` pipeline the best case: recorded Bottle Toss positions with no prediction error, sampled at 10 to 200 Hz. Its interpolating cubic B-spline matches position to 0.03 to 0.04 rad beyond 20 Hz, yet the differentiated velocity stays 0.8 to 1.8 rad/s from the recording and drifts upward with the rate, which is consistent with the first-difference form in Sec. 3.2.

6 Limitations and Future Directions

As a first study of velocity as a supervised VLA output, we fixed some choices to isolate the question. We train on joint velocities recorded with the demonstrations; for datasets that log only positions, our sampling-rate study (Sec. 5.4) characterizes what a differentiated fit recovers, and treating velocity as a latent, supervised where available and inferred elsewhere (Ye et al., 2025), is the natural next step. We evaluate on tasks that velocity decides; the smaller gains on the two timing tasks and on a timing-aware backbone (Sec. 5.3) mark the boundary, and since the velocity expert can output zero, we expect VIPER to reduce to the base policy on quasi-static tasks, which is easy to verify.
Acceleration, torque, and contact force describe the same motion as velocity, and each is a candidate for a further expert on the same backbone; our coupling result suggests the open question is holding such outputs at a common phase, a structured-generation problem the flow-matching community is well placed to take up (Lipman et al., 2022). Real-world online RL (Dong et al., 2026) to correct velocity targets is a second direction, since velocity is what a reward on momentum or timing most directly shapes. We commit to releasing code, demonstrations with joint velocities, and the RoboTwin task suite so that more seeds, backbones, and dynamic tasks can be added by anyone.

AI USE STATEMENT

In this work, we used generative AI for summarizing relevant background literature during the literature survey, polishing the wording of the manuscript for better readability/clarity and checking for inconsistencies in the writing. We have not used generative AI tools to generate synthetic datasets or to provide critical ingredients for mathematical analysis.. We have reviewed all AI-assisted work by cross-checking all source materials referenced by AI, and additional human passes on the writing to make sure the meaning is not misrepresented. We take full responsibility for the final content of this work, including text, claims, or artifacts produced with the help of generative AI.

ETHICS STATEMENT

Since the two dynamic tasks, Drop Catch and Bottle Handover, involve human subjects, the YAM arm is safeguarded with a range limit to prevent erroneous model output from injuring human subjects. The human subject's torso is kept outside of this range limit at all times during testing. Additionally, as a fail-safe mechanism, the robot motor is set to a motor back-off force, such that if the motor senses resistance past a threshold, it will not take any further commands in that direction. For data release, we will anonymize any identifiable information of the human subject before publishing to the general public.

REPRODUCIBILITY STATEMENT

We acknowledge the importance of reproducibility in research as a great way to benefit the broader ML community, as an effort towards this direction, we will be releasing the codebase used in our paper to produce success rates from our paper. Additionally, we plan to open-source the data and sim environment along with the data preprocessing code, so others can try out our tasks. All such information will be available shortly after submission on our project website at https://viper-robot.github.io/. Appendix C states the idealized alignment analysis and the assumptions under which it holds.

REFERENCES

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. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

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. *arXiv preprint arXiv:2212.06817*, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

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. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Perry Dong, Kuo-Han Hung, Dorsa Sadigh, and Chelsea Finn. Reinforcement learning for real-time vision-language-action policies. *arXiv preprint arXiv:2609.18207*, 2026.

Zhenyang Feng, Jimin Heo, Erik B. Sudderth, and Unnat Jain. Tempo: Learning temporal context for dynamic robot manipulation, 2026. URL `https://arxiv.org/abs/2609.16864`.

Yanzhao Guo, Wenkai Chen, and Jianwei Zhang. Reactvla: Fast and lightweight reactive robot manipulation via improved mean flow action generation. *arXiv preprint arXiv:2606.14255*, 2026.

Xiaoshen Han, Haoyu Xiong, Haonan Chen, Chaoqi Liu, Antonio Torralba, Yuke Zhu, and Yilun Du. B-spline policy: Accelerating manipulation policies via b-spline action representations. *arXiv preprint arXiv:2607.09648*, 2026.

Johannes Hechtl, Philipp Schmitt, Georg von Wichert, and Wolfram Burgard. Enabling dynamic tracking in vision-language-action models via time-discrete and time-continuous velocity feed-forward. *arXiv preprint arXiv:2603.16218*, 2026.

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*, 2024.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

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

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. Rdt-1b: a diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, volume 2025, pp. 29982–30009, 2025.

Yuxiang Lu, Zhe Liu, Xianzhe Fan, Zhenya Yang, Jinghua Hou, Junyi Li, Kaixin Ding, and Hengshuang Zhao. Faster: Rethinking real-time flow vlas. *arXiv preprint arXiv:2603.19199*, 2026.

Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. Serl: A software suite for sample-efficient robotic reinforcement learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16961–16969. IEEE, 2024.

Khang Nguyen, An T Le, Tien Pham, Manfred Huber, Jan Peters, and Minh Nhat Vu. Flowmp: Learning motion fields for robot planning with conditional flow matching. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11291–11297. IEEE, 2025.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, 2024.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Physical Intelligence, Bo Ai, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Greg Balke, Kevin Black, George Bokinsky, Shihao Cao, Thomas Charbonnier, et al. $\pi_{0.7}$: A steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

PI, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Daniel San José Pro, Oliver Hausdörfer, Ralf Römer, Maximilian Dösch, Martin Schuck, and Angela P Schoellig. Crisp-compliant ros2 controllers for learning-based manipulation policies and teleoperation. *IEEE Robotics and Automation Practice*, 2026.

Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *2009 IEEE international conference on robotics and automation*, pp. 489–494. IEEE, 2009.

John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pp. 1–10. Berlin, Germany, 2013.

Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. Leave no observation behind: Real-time correction for vla action chunks. *arXiv preprint arXiv:2509.23224*, 2025.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. In *International Conference on Learning Representations*, volume 2026, pp. 18567–18602, 2026.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Shahram Najam Syed, Arthur Jakobsson, Haoran Hao, and Jeffrey Ichnowski. Intercepting the future: Latent-space predictive world model for dynamic vla manipulation. *arXiv preprint arXiv:2606.02486*, 2026.

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.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

Junjie Wen, Yichen Zhu, Minjie Zhu, Zhibin Tang, Jinming Li, Zhongyi Zhou, Xiaoyu Liu, Chaomin Shen, Yaxin Peng, and Feifei Feng. Diffusionvla: Scaling robot foundation models via unified diffusion and autoregression. In *ICML*, 2025.

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.

Fan Yang, Peiguang Jing, Kaihua Qu, Ningyuan Zhao, and Yuting Su. Abpolicy: Asynchronous b-spline flow policy for real-time and smooth robotic manipulation. *arXiv preprint arXiv:2602.23901*, 2026.

Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Se June Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, et al. Latent action pretraining from videos. In *ICLR*, 2025.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 60580–60606, 2026.