THE LOW-RANK STRUCTURE OF VLA REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) is increasingly used to post-train vision-language-action (VLA) models, yet how RL reshapes these policies remains poorly understood. We find that RL across widely used flow-based VLA models, including $\pi_{0.5}$ and GR00T N1.5/N1.6, on LIBERO, ManiSkill, MetaWorld, and CALVIN induces substantially lower-rank parameter updates that are highly concentrated in the action expert's Timestep Modules, a small and previously overlooked component. Through systematic module-replacement experiments, we further show that these modules capture a disproportionate share of the performance gains from RL: in $\pi_{0.5}$, keeping only their RL updates yields performance within 1.6% of the full RL policy despite comprising only 27.6% of the action expert's parameters. We then characterize what is encoded in these Timestep Modules. First, we show that RL specializes them to the discrete denoising timesteps used during rollouts, and that this discrete-timestep training underlies the low-rank updates. Second, we find that among their outputs, the shift vector changes most distinctly under RL, and through probing, we show that shift update directions strongly predict task success (ROC-AUC up to 99.6%). Third, we find that the geometry of shift updates reflects task relationships, as their pairwise similarity correlates with cross-task transfer patterns. Building on these findings, we show that steering along shift update directions further improves RL-trained policies without additional RL training. Overall, we provide a systematic understanding of how RL reshapes VLA policies by studying how learned signals are encoded in parameter space, offering insights into more efficient and interpretable VLA post-training.

1 INTRODUCTION

Vision-language-action (VLA) models have emerged as promising general-purpose robot foundation models, demonstrating strong performance across diverse manipulation tasks and embodiments (Zitkovich et al., 2023; O'Neill et al., 2024; Black et al., 2024; Bjorck et al., 2025). Similar to other foundation models, VLAs are typically trained in two stages: large-scale pre-training on vision-language and action data, followed by post-training on smaller, task-specific datasets (Kim et al., 2025; Black et al., 2024). As collecting high-quality robot demonstrations is often costly, recent work has demonstrated the effectiveness of reinforcement learning (RL) for VLA post-training in simulation (Tan et al., 2025; Li et al., 2026; Chen et al., 2025; Wang et al., 2026b), building on the success of RL in LLM post-training (Ouyang et al., 2022; Guo et al., 2025). For LLMs, extensive work has characterized the parameter-space structure induced by RL post-training, enabling more effective and efficient learning strategies, including low-rank methods such as LoRA (Schulman & Lab, 2025; Cai et al., 2026; Zhang et al., 2026b; Yin et al., 2026). Yet, an analogous understanding of VLA RL remains largely unexplored, particularly given the distinct flow-based action expert architectures commonly used in modern VLAs (§ 2). Consequently, it remains unclear where and how RL learning signals are encoded in the parameter space of VLAs.

In this work, we seek to systematically understand how RL learning signals reshape VLAs from a parameter-space perspective. Across widely used flow-based VLA families (π, GR00T) (Intelligence et al., 2025; Bjorck et al., 2025) and manipulation benchmarks (LIBERO, ManiSkill, MetaWorld, CALVIN) (Liu et al., 2023; Tao et al., 2024; Yu et al., 2020; Mees et al., 2022), we consistently find that RL induces low-rank parameter updates concentrated in a small, specific module. Surprisingly, the dominant RL updates are concentrated in a previously overlooked component of

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.

the action expert, the *Timestep Modules* (Figure 1), which condition flow-matching policies on the denoising timestep. Despite exhibiting strongly low-rank updates and thus being natural targets for parameter-efficient adaptation, these modules are typically omitted from standard LoRA configurations. Furthermore, we do not observe the same low-rank structure in flow-based image and video generation models trained with RL, suggesting that this phenomenon is specific to action policies rather than a generic property of flow-based architectures (§ 3.1). Through controlled module-replacement experiments, we show that the Timestep Modules account for a disproportionate share of the performance gains from RL. Consistent with this finding, targeting LoRA specifically to the Timestep Modules outperforms standard LoRA configurations (§ 3.2).

Building on these findings, we further characterize how the individual components of the Timestep Modules encode RL learning signals. First, unlike standard behavior cloning, RL sharply specializes the Timestep Modules to selectively respond to the discrete denoising timesteps encountered during on-policy learning, and this discrete-timestep training underlies the low-rank updates (§ 4.1). Second, the learned updates encode task-specific information in a highly compact form due to their low-rank structure, effectively collapsing into a small number of vectors. As a result, one output of the Timestep Modules—the shift vector—carries most of this task-related information (§ 4.2). We confirm this through probing and task correlation analyses: shift update directions predict downstream task success with an ROC-AUC of up to 99.6%, and their similarity patterns correlate strongly with cross-task transfer patterns, with Spearman $\rho = 0.794$ (§ 5.1). Finally, since RL induces a distinct low-dimensional update direction for each task, steering RL-trained policies along these directions further improves performance at test time (§ 5.2). Together, we find that the Timestep Modules encode RL learning signals as a small set of timestep-specific vectors that are independent of the image and language inputs or the current task progress, and that their updates alone are sufficient to capture a substantial portion of the performance gains from RL.

Overall, we provide a parameter-level characterization of VLA post-training, showing, to our knowledge for the first time, that RL induces low-rank, structured parameter updates concentrated in the Timestep Modules, driven by training on discrete denoising timesteps. More broadly, our findings lay the groundwork for future work on more parameter-efficient RL post-training, improved adaptation strategies, and the composition and transfer of task-specific RL behaviors.

2 Background

2.1 Flow-based VLA Policies

Recent VLAs increasingly pair a pretrained vision-language backbone with a dedicated flow-based action expert for continuous robot control (Black et al., 2024; Intelligence et al., 2025; Bjorck et al., 2025). These policies commonly generate *action chunks*, improving temporal consistency (Zhao et al., 2023; Chi et al., 2025) and reducing generation latency compared with discrete action-token policies (Black et al., 2024). Such action experts are trained with a flow-matching objective (Lipman et al., 2023) to predict the transformation from a noisy action chunk toward the demonstrated action chunk. The standard behavior cloning (BC) flow-matching objective for training the action expert V_θ, parameterized by θ, is:

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{a,\epsilon,\tau}\left[\|V_\theta(a_\tau, \tau, o) - (a - \epsilon)\|_2^2\right], \qquad a_\tau = (1-\tau)\epsilon + \tau a, \tag{1}$$

where a is the demonstrated action chunk, ϵ is sampled noise, and $\tau \sim \mathcal{U}(0,1)$ is the denoising timestep. Because τ is sampled continuously, the action expert is trained across the full interval $\tau \in [0,1]$. At inference time, the action expert is run at a discrete sequence of timesteps, iteratively transforming an initial noise sample into the final action chunk.

Timestep Modules. In this work, we use *Timestep Modules* to refer to the components of the action expert that transform the scalar denoising timestep τ into vector embeddings that modulate the hidden states. For example, a three-step denoising process uses $\tau \in \{0, \frac{1}{3}, \frac{2}{3}\}$ as inputs. In the widely used $\pi_{0.5}$ model, the Timestep Modules consist of a Time MLP and adaptive RMS normalization (AdaRMS) (Figure 1) (Intelligence et al., 2025), which together account for only 27.58% of the action expert's parameters, with most (98.23%) belonging to AdaRMS. During each forward pass, the Time MLP first maps a sinusoidal timestep embedding $\phi(\tau)$ to a conditioning vector c_τ:

$$c_\tau = \mathrm{TimeMLP}(\phi(\tau)). \tag{2}$$

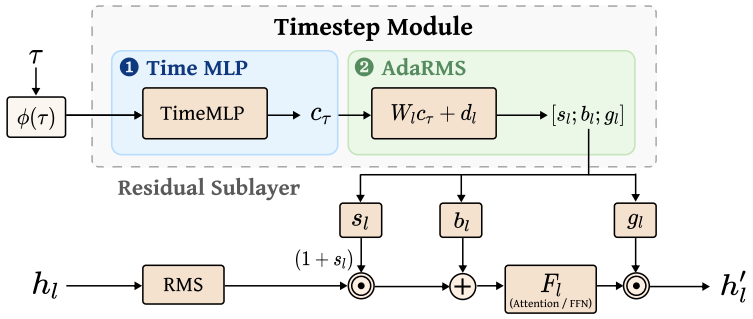


Figure 1: **Overview of the Timestep Module in $\pi_{0.5}$.** The timestep embedding is transformed by a TimeMLP and AdaRMS to produce timestep-dependent scale, shift, and gate vectors that modulate each attention or FFN residual sublayer.

AdaRMS, parameterized by W_ℓ and d_ℓ, then maps c_τ to scale, shift, and gate vectors (s_ℓ, b_ℓ, g_ℓ):

$$[s_\ell; b_\ell; g_\ell] = W_\ell c_\tau + d_\ell, \tag{3}$$

$$z_\ell = (1 + s_\ell) \odot \mathrm{RMS}(h_\ell) + b_\ell, \tag{4}$$

$$h'_\ell = h_\ell + g_\ell \odot F_\ell(z_\ell). \tag{5}$$

Here, the hidden state h_ℓ is RMS-normalized, scaled, and shifted to obtain z_ℓ, which is passed through the corresponding attention or feed-forward sublayer F_ℓ and gated before being added back to the residual stream. We collectively refer to the Time MLP and the AdaRMS parameters $\{W_\ell, d_\ell\}$ as the Timestep Modules. Notably, the resulting scale, shift, and gate vectors are conditioned solely on the timestep τ, independent of visual or language inputs. Consequently, with a fixed schedule of K denoising steps, the Timestep Modules produce only K distinct sets of modulation vectors per layer at inference. We further detail the Timestep Modules of various VLAs in § A.

2.2 Reinforcement Learning for VLA

Recent work has shown that reinforcement learning (RL) can effectively post-train VLAs through online interaction in simulation, reducing reliance on additional expert demonstrations (Li et al., 2026; Chen et al., 2025). Following recent work on flow-based VLA RL (Chen et al., 2025), we use proximal policy optimization (PPO) (Schulman et al., 2017) as the base RL algorithm:

$$\mathcal{L}_{\mathrm{PPO}}(\theta) = -\mathbb{E}_i \left[\min \left(\rho_i(\theta) \hat{A}_i,\ \mathrm{clip}\left(\rho_i(\theta), 1-\varepsilon, 1+\varepsilon\right) \hat{A}_i \right) \right], \tag{6}$$

where $\rho_i(\theta)$, $\hat{A}_i$, and ε are the policy ratio, estimated advantage, and clipping threshold, respectively. Notably, on-policy RL such as PPO performs updates on trajectories generated by the action expert and thus trains the action expert only at the discrete denoising timesteps used during rollouts. This differs from BC, which trains the action expert on continuous timesteps in $[0, 1]$ (§ 2.1).

3 Parameter Updates during VLA Reinforcement Learning

We begin by examining where and how RL updates the parameters of VLAs. By analyzing the density and effective rank of parameter updates across multiple VLA architectures and benchmarks, we find that (1) RL induces low-rank updates concentrated in the Timestep Modules, an overlooked component of the action expert (§ 3.1), and (2) these low-rank updates account for a disproportionate share of the performance gain from RL, as shown through controlled module-replacement experiments (§ 3.2).

3.1 Where and How Does RL Update VLA Parameters

Settings. We study both publicly released and in-house-trained BC and RL checkpoints from multiple flow-based VLA families, including $\pi_{0.5}$, GR00T N1.5, GR00T N1.6, and, additionally, SmolVLA (Intelligence et al., 2025; Bjorck et al., 2025; Shukor et al., 2025), spanning LIBERO,

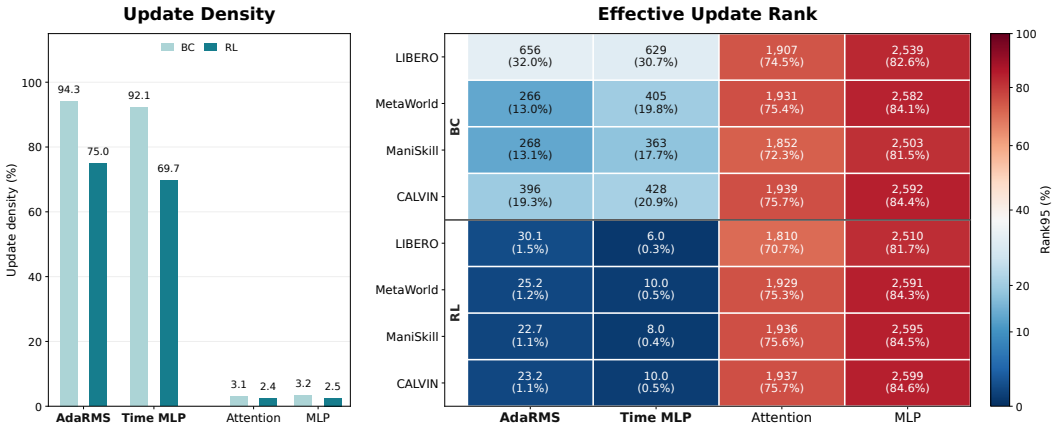


Figure 2: **Timestep Modules receive dense but low-rank RL updates.** (Left) Update density for BC and RL checkpoints trained on LIBERO-Spatial from the same $\pi_{0.5}$ policy. (Right) Effective update rank across $\pi_{0.5}$ BC and RL checkpoints.

ManiSkill, MetaWorld, and CALVIN (Liu et al., 2023; Tao et al., 2024; Yu et al., 2020; Mees et al., 2022) (§ B.1).

We define the parameter update as $\Delta W = W_{\text{trained}} - W_{\text{reference}}$, where W_{trained} and $W_{\text{reference}}$ denote the parameters after and before training, respectively. We define *update density* as the fraction of parameters whose absolute change exceeds a small threshold (10^{-5}), following Mukherjee et al. (2025), and define the *effective update rank* as the smallest number of singular directions that explain 95% of the update's squared Frobenius norm:

$$r_{95}(\Delta W) = \min\left\{ r : \frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_i \sigma_i^2} \geq 0.95 \right\}, \tag{7}$$

where σ_i are the singular values of ΔW in descending order. We further ablate thresholds in § C.1.

Results. Figure 2 (left) shows the update density of each component for BC and RL checkpoints trained on LIBERO-Spatial with $\pi_{0.5}$. In both cases, the Timestep Modules (AdaRMS and Time MLP) are updated far more densely (70–95%) than attention and MLP modules ($< 5\%$). Figure 2 (right) shows the effective rank of the updates across multiple BC and RL checkpoints. Here, BC and RL differ qualitatively, as RL updates to the Timestep Modules have an effective rank of only 6–30, roughly an order of magnitude lower than BC (250–650), whereas RL updates to attention and MLP modules remain high-rank. We provide additional results in § C.2.

Overall, we find that VLA post-training concentrates its updates in the Timestep Modules, and RL further compresses these updates to a low effective rank. Furthermore, this low-rank structure does not appear in RL-trained image and video DiTs, whose Timestep Module updates require 29–80% of the available rank (§ C.3), suggesting that it is specific to VLA policies rather than a general property of timestep-conditioned models.

Notably, the Timestep Modules produce only the scale, shift, and gate vectors that modulate the hidden states (Eq. 3), and these vectors depend solely on the denoising timestep τ. A low-rank update therefore shifts these modulation vectors along only a few fixed directions, regardless of the observation, instruction, or current stage of task execution. This raises the question of whether such simple, input-independent changes can account for the performance gains from RL, which we test next through module replacement.

Takeaway 1: RL learns dense, low-rank updates in VLA Timestep Modules.

Across various flow-based VLAs, RL updates the Timestep Modules more densely than other modules and, unlike BC, induces low-rank updates to the Timestep Modules.

Table 1: **Timestep Modules encode more of the RL gain than all other parameters.** Success rates (%) by condition. LIBERO is averaged over the Spatial, Object, and Goal suites. †GR00T N1.6 is evaluated on LIBERO-Spatial only. Best results are in **bold**, second-best results are underlined, and Δ rows show the difference from RL.

	$\pi_{0.5}$				GR00T N1.5	GR00T N1.6	
Condition	**LIBERO**	**ManiSkill**	**MetaWorld**	**CALVIN**	**LIBERO**	**LIBERO†**	**Avg.**
Base	87.7	42.2	42.8	61.8	54.4	75.6	60.8
RL	**96.5**	**89.1**	69.2	**87.7**	**90.3**	**85.0**	**86.3**
Timestep Modules only	95.1	87.5	**69.6**	87.0	61.3	84.8	80.9
Δ vs. RL	(−1.4)	(−1.6)	(+0.4)	(−0.7)	(−29.0)	(−0.2)	(−5.4)
MLP+Attn only	90.5	49.4	54.6	62.7	57.9	84.6	66.6
Δ vs. RL	(−6.0)	(−39.7)	(−14.6)	(−25.0)	(−32.4)	(−0.4)	(−19.7)
Timestep Modules Size	27.58%				14.83%	16.50%	—

3.2 Do Timestep Modules Capture the RL Gain?

Settings. Building on Takeaway 1, we test whether the Timestep Modules actually account for the performance gains from RL. Using the RL-trained $\pi_{0.5}$, GR00T N1.5, and GR00T N1.6 checkpoints from § 3.1, we perform module-replacement experiments. Starting from each RL checkpoint, we either (1) replace the attention and FFN parameters with their base parameters, keeping only the Timestep Modules RL-trained (*Timestep Modules only*), or (2) replace the Timestep Modules with their base parameters, keeping all other RL-trained parameters fixed (*MLP+Attn only*).

Results. As shown in Table 1, *Timestep Modules only* consistently preserves more of the RL gain than *MLP+Attn only*, despite the Timestep Modules comprising only 14.83–27.58% (§ A) of the parameters. Notably, in $\pi_{0.5}$, *Timestep Modules only* nearly matches the full RL policy, reaching 69.6% on MetaWorld and falling behind by only 1.4%, 1.6%, and 0.7% on LIBERO, ManiSkill, and CALVIN, respectively, whereas *MLP+Attn only* drops by up to 39.7% on ManiSkill.

We further verify this with two additional experiments. First, under the *Timestep Modules only* setting, reconstructing the Timestep Module updates from only their top four singular directions retains most of the performance, while removing these directions largely degrades it, highlighting that the low-rank updates carry most of the RL gain (§ C.4). Second, we compare LoRA on the Timestep Modules against LoRA on all other modules, the standard setting, and find that Timestep-targeted LoRA converges faster and to a higher success rate (§ C.5).

Overall, these results indicate that a large portion of the performance gain from RL is captured by low-rank updates to the Timestep Modules. Since these modules only produce the scale, shift, and gate vectors, which depend solely on the denoising timestep and not on the image, instruction, or task progress, this suggests that much of what RL learns is a simple, timestep-specific change to how the action expert is modulated at each denoising step.

Takeaway 2: Timestep Modules encode a disproportionately large share of the RL gain.

Through controlled module-replacement experiments, we show that RL encodes much of its gain in the Timestep Modules, largely through low-rank updates.

4 What Do the Timestep Modules Encode?

In Takeaways 1 and 2, we showed that RL updates to the Timestep Modules exhibit a low-rank structure that captures a substantial portion of the RL gain. In this section, we examine AdaRMS, the main component of the Timestep Modules: (1) how it specializes to the denoising timesteps under RL, which underlies the low-rank updates (§ 4.1), and (2) how the scale, shift, and gate vectors change under RL, with the shift vectors standing out (§ 4.2).

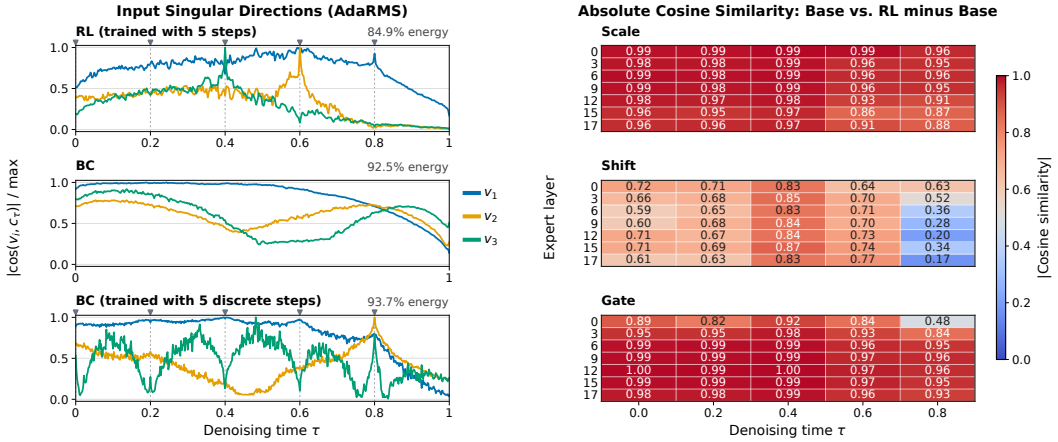


Figure 3: **AdaRMS updates specialize to denoising timesteps, largely changing the shift vector.** (Left) Top three input singular directions of LIBERO-Spatial AdaRMS updates for BC and RL. Unlike standard BC, RL and discrete-timestep BC checkpoints peak near the denoising timesteps used during training. (Right) Absolute cosine similarity between the base AdaRMS output and the RL-induced change (RL − base) in MetaWorld for the scale, shift, and gate vectors, respectively.

4.1 Specialization to Discrete Timesteps

Settings. As discussed in § 2.1, the Timestep Modules take only the denoising timestep as input, without any image, text, or task progress. We therefore analyze how AdaRMS learns to respond across denoising timesteps under standard BC and RL, and additionally under BC trained only on the discrete timesteps used during RL (*discrete-timestep BC*) for comparison. Specifically, we compute the SVD of each AdaRMS update ΔW_ℓ to extract the top three input singular directions v_i, which account for most of the update energy. We then measure the normalized absolute cosine similarity between each v_i and the conditioning vector c_τ across denoising timesteps to examine how the AdaRMS update responds to each timestep.

Results. Figure 3 (left) shows the results for $\pi_{0.5}$ on LIBERO-Spatial under standard BC, RL, and discrete-timestep BC. Under standard BC, the similarity varies smoothly across timesteps, whereas RL produces sharply localized responses around the discrete timesteps used during training. Interestingly, this matches the difference in training discussed in § 2.2, as BC trains on all timesteps $\tau \in [0, 1]$, while on-policy RL trains only on the fixed timesteps used during rollouts. Furthermore, discrete-timestep BC exhibits similarly localized responses, and its updates are likewise low-rank (§ C.2), indicating that discrete-timestep training is the main cause of the low-rank structure. We provide additional results in § C.6.

This suggests that RL learns timestep-specific modulation patterns for the discrete timesteps used during rollouts. Since the scale, shift, and gate vectors produced by the Timestep Modules capture much of the RL gain (Takeaway 2), the AdaRMS update only needs to act on a few distinct timesteps, for which a low-rank update suffices.

Takeaway 3: RL specializes AdaRMS to the discrete denoising timesteps.

Unlike standard BC, RL trains AdaRMS only at the few timesteps used during rollouts, causing the module to specialize to those timesteps and resulting in low-rank updates.

4.2 Disentangling Scale, Shift, and Gate

Settings. We examine how the three AdaRMS output vectors (scale, shift, and gate) change due to RL. We compute the absolute cosine similarity between each output's base value and its RL-induced change (RL − base) at each denoising timestep used during RL training. Values near 1 indicate rescaling along the existing direction, while values near 0 indicate a new direction.

Table 2: **Linear probing of shift vectors in $\pi_{0.5}$.** F1 and ROC-AUC (%) for probes applied to the base and RL policies. Random denotes probes trained with shuffled outcome labels.

		Base Policy		RL Policy	
Benchmark	Condition	F1	AUC	F1	AUC
LIBERO-Spatial	**Ours**	**97.1**	**99.6**	**96.3**	**96.6**
	Random	58.0 ± 20.5	46.5 ± 4.0	75.2 ± 13.6	51.5 ± 3.9
LIBERO-Object	**Ours**	**98.2**	**98.6**	**97.3**	**97.4**
	Random	76.3 ± 10.2	53.4 ± 6.3	86.2 ± 15.8	47.2 ± 13.3
LIBERO-Goal	**Ours**	**77.5**	**98.9**	**74.9**	**68.0**
	Random	47.6 ± 37.2	49.0 ± 3.5	65.4 ± 28.9	52.6 ± 6.4
ManiSkill	**Ours**	**95.8**	**99.6**	**98.3**	**98.3**
	Random	42.9 ± 8.9	51.8 ± 7.0	87.1 ± 3.4	46.5 ± 9.3
MetaWorld	**Ours**	**58.3**	**74.2**	**81.1**	**90.1**
	Random	43.4 ± 7.8	49.4 ± 4.8	64.8 ± 6.6	49.4 ± 6.1

Results. As shown in Figure 3 (right), the RL-induced change is substantially more aligned with the base output for scale and gate vectors than for shift vectors, and varies more across denoising timesteps than across Transformer sublayers. This suggests that RL largely rescales the existing scale and gate patterns while introducing new directions predominantly in the shift vectors, which may therefore encode more information learned through RL.

5 Understanding the Shift Vector

We find that RL induces low-rank updates concentrated in the Timestep Modules, with the shift vector changing most among their outputs. In this section, we examine what information these shift updates capture (§ 5.1) and whether they can be exploited to steer task outcomes (§ 5.2).

5.1 What Do the Shift Updates Encode?

5.1.1 Do Shift Updates Encode Task Outcomes?

Settings. Because RL improves task success partly through shift updates that are directly added to the hidden state, we hypothesize that the shift update directions capture success-relevant information in the representation space. This connects naturally to prior probing work in LLMs, which projects hidden representations onto specific directions to reveal answer correctness (Zhang et al., 2025; Cencerrado et al., 2026), as well as recent VLA work on task success prediction (Gu et al., 2025; Zhang et al., 2026a). Following this perspective, we use each RL-induced shift update, $\Delta b = b_{\mathrm{RL}} - b_{\mathrm{base}}$, as a probing direction in representation space and test whether hidden-state projections onto this direction predict the episode's eventual success or failure.

For each checkpoint, we first compute a separate shift update for every attention or MLP sublayer–timestep pair. During rollouts, we extract the hidden representation at each corresponding sublayer input, project it onto $\Delta\mathbf{b}$, and average the projection scores over the episode. The resulting episode-level scores across all sublayer–timestep pairs form a feature vector, which an ℓ_2-regularized logistic regression probe combines across sublayers and timesteps to predict episode success or failure.

We evaluate both the base and RL $\pi_{0.5}$ policies on LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, ManiSkill, and MetaWorld. Within each benchmark, tasks are split 70:30 into training and test sets, and the probe is trained on the training tasks and evaluated on held-out tasks.

Results. As shown in Table 2, shift update projections strongly predict episode outcomes, reaching ROC-AUCs of 98.6–99.6 across LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and ManiSkill, well above random-label controls. Notably, our probes are predictive for both the base and RL policies, suggesting that RL aligns with success-relevant structure already present in the representation space and preserves this alignment after optimization. Even a single sublayer–timestep direction yields ROC-AUCs of 96.9–99.0 on LIBERO-Spatial, LIBERO-Object, and ManiSkill. Although

the sign of the association varies across sublayer–timestep pairs, predictive performance remains consistently strong (§ C.8). Overall, these results suggest that RL-induced shift updates are structured around outcome-relevant directions in the representation space.

5.1.2 Does Shift Update Geometry Encode Task Relationships?

Settings. We next ask whether shift update geometry reflects relationships across tasks. We hypothesize that related tasks exhibit aligned shift updates and similar cross-task transfer behavior. We train a separate RL policy for each of the ten LIBERO-Spatial tasks and construct two vectors for each task: a *shift-update vector* from its single-task RL policy and a *transfer-effect vector* containing the success-rate gains of all ten policies over the base policy on that task. We then compare tasks using each of these two vectors. For shift-update vectors, we measure pairwise alignment using absolute cosine similarity, since vectors pointing in opposite directions can still encode similar task-related information. For transfer-effect vectors, we measure pairwise similarity using Spearman correlation. Finally, for each task, we compute the Spearman correlation between its shift-update-vector similarities and transfer-effect-vector similarities to the other tasks. This tests whether tasks with geometrically aligned shift updates also exhibit similar cross-task transfer behavior.

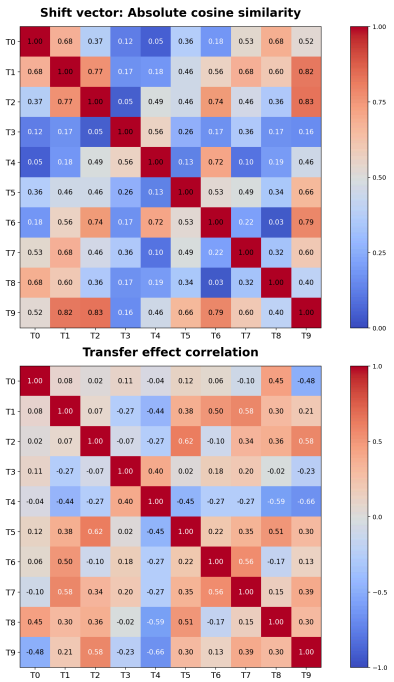


Figure 4: **Cross-task structure in $\pi_{0.5}$ on LIBERO-Spatial subtasks.** (Top) Absolute cosine similarity between shift vectors. (Bottom) Correlation of cross-task transfer effects.

Results. We find that shift-update alignment is positively associated with cross-task transfer behavior, with a mean task-wise correlation of 0.504 (range: 0.224–0.794). Figure 4 further shows that the pairwise structure of shift-update alignment (top) closely resembles that of cross-task transfer similarity (bottom). These results suggest that the geometry of shift update directions reflects relationships among the tasks on which they are learned, providing a compact representation of how RL updates may transfer across tasks.

5.2 Exploiting the Shift Updates

Settings. Having shown that shift updates encode task outcomes, we ask whether this information can be exploited to improve policy performance. Specifically, we steer representations along the shift update direction toward values associated with successful episodes. We select the sublayer–timestep pair with the highest probing F1 score and perform steering along its shift update direction. Inspired by prior adaptive activation steering methods for LLMs (Cheng & Alonso, 2026; Park et al., 2026), we intervene only when the probe indicates that the projection deviates from the mean projection of successful episodes, shifting it by the minimum amount needed to reach this mean. We use the same models, benchmarks, and split as in § 5.1.1.

Table 3: **$\pi_{0.5}$ adaptive steering results.** Success rates (%) on the 30% test split for RL, random vector steering, and shift vector steering.

Checkpoint	RL	Random	Steered
LIBERO-Spatial	86.7	87.2 ± 0.3	**90.0** (+3.3)
LIBERO-Object	94.7	94.7 ± 0.0	**96.7** (+2.0)
LIBERO-Goal	90.7	90.7 ± 0.0	**94.7** (+4.0)
ManiSkill	90.0	88.4 ± 3.4	**92.0** (+2.0)
MetaWorld	66.7	64.1 ± 1.8	**68.7** (+2.0)

Results. As shown in Table 3, adaptive steering improves the final RL policy across all five benchmarks by 2.0–4.0% and consistently outperforms matched-norm random directions. For example, success rates increase from 86.7% to 90.0% on LIBERO-Spatial and from 90.7% to 94.7% on LIBERO-Goal. Similar gains from steering the base policy further support the causal role of the

learned shift directions (§ C.9). Overall, these results suggest that the shift updates capture an actionable control signal that is not fully exploited by the final RL policy and can be further leveraged at inference time to improve an already trained policy.

Takeaway 4: Shift updates largely encode task-relevant RL signals.

Shift updates encode task outcomes and cross-task relationships, enabling test-time steering to further improve the final RL policy.

6 Related Work

6.1 Vision-Language-Action Models

Vision-language-action models build on pretrained vision-language models to map visual observations and language instructions to robot actions (Zitkovich et al., 2023; O'Neill et al., 2024; Kim et al., 2025). Recent VLAs increasingly adopt dedicated diffusion- or flow-based action experts (Black et al., 2024; Bjorck et al., 2025; Intelligence et al., 2025), which, combined with action chunking, generate sequences of actions jointly, improving temporal coherence and inference latency (Zhao et al., 2023). In this work, we focus on flow-based VLAs and investigate how RL post-training reshapes their action experts, particularly the Timestep Modules. Our work is closely related to recent efforts to understand VLA internal representations through probing and representation steering (Häon et al., 2025; Wang et al., 2026a; Gu et al., 2025; Buurmeijer et al., 2026), but differs in focusing on how RL reshapes these representations.

6.2 Reinforcement Learning

Inspired by the success of RL in LLMs (Ouyang et al., 2022; Guo et al., 2025), recent work has successfully post-trained VLAs through online RL in simulation (Tan et al., 2025; Li et al., 2026; Liu et al., 2025b). In the LLM RL literature, rigorous studies have examined RL at the parameter level, identifying sparse parameter updates (Mukherjee et al., 2025) and, in other settings, dominant low-rank update directions (Cai et al., 2026). Other work shows that RL can succeed by updating only a few parameters, motivating more parameter-efficient post-training (Morris et al., 2026; Yin et al., 2026; Schulman & Lab, 2025; Zhang et al., 2026b). While recent work has proposed various recipes for optimizing flow-based policies (Liu et al., 2025a; McAllister et al., 2026), including the action experts of recent VLAs (Chen et al., 2025), what exactly happens to the action expert during RL has not yet been studied, which we address in this work.

7 Conclusion and Discussion

In this work, we studied how RL reshapes flow-based VLAs from a parameter-space perspective across $\pi_{0.5}$ and GR00T on LIBERO, ManiSkill, MetaWorld, and CALVIN. We consistently find, to our knowledge for the first time, that RL encodes its learning signal in low-rank updates to the Timestep Modules, a small and previously overlooked component of the action expert. Leveraging this structure, we further show through probing and steering experiments that much of the RL signal is captured at the vector level and concentrated in one of their outputs, the shift vector. Together, our results suggest that much of RL post-training in VLAs can be explained by simple, timestep-specific vector-level modulations of the action expert's hidden states.

Our findings primarily motivate more parameter-efficient RL post-training and suggest several directions for future work. First, since discrete-timestep BC induces a similar low-rank structure (§ 4.1), investigating its effects and potential advantages over standard continuous-timestep BC is a promising direction for VLA policies. Second, our results may help explain the effectiveness of residual RL (Johannink et al., 2019) and related approaches that adapt frozen VLAs through lightweight trainable modules (Xiao et al., 2026; Wagenmaker et al., 2025; Xu et al., 2026). If full-parameter RL largely reduces to low-rank modulation, a small residual module may suffice to capture much of its benefit, potentially informing future residual RL designs.

AI USE STATEMENT

In this work, we used generative AI assistants to edit the manuscript for clarity and grammar, format tables and LaTeX code, prototype figures, and assist with parts of our analysis and plotting code. We did not use generative AI tools to propose or refine hypotheses, design or provide feedback on research methods or experiments, or interpret results. Generating synthetic datasets, developing theoretical models or conceptual frameworks, formulating mathematical claims, proving mathematical claims or assisting with the writing of proofs, translation, dataset cleaning or reformatting, and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to propose a title or keywords, identify relevant literature, and source or search for information. The authors reviewed and rewrote all AI-assisted text and reviewed all AI-assisted code for correctness. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We plan to publicly release the code used in our experiments, along with relevant artifacts such as our trained checkpoints, upon acceptance. To support future reproduction, we provide the detailed hyperparameters and settings used throughout our experiments in § B.

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. URL https://arxiv.org/abs/2503.14734.

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. URL https://arxiv.org/abs/2410.24164.

Hugo Buurmeijer, Carmen Amo Alonso, Aiden Swann, and Marco Pavone. Observing and controlling features in vision-language-action models. In *Mechanistic Interpretability Workshop at ICML 2026*, 2026. URL https://openreview.net/forum?id=rkloeJAWnz.

Yuchen Cai, Ding Cao, Xin Xu, Zijun Yao, Yuqing Huang, Zhenyu Tan, Benyi Zhang, Guangzhong Sun, Guiquan Liu, and Junfeng Fang. On predictability of reinforcement learning dynamics for large language models. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=SdHmA6BYVJ.

Iván Vicente Moreno Cencerrado, Arnau Padrés Masdemont, Anton Gonzalvez Hawthorne, David Demitri Africa, and Lorenzo Pacchiardi. No answer needed: Predicting LLM answer accuracy from question-only linear probes. In *ICLR 2026 Workshop on Principled Design for Trustworthy AI - Interpretability, Robustness, and Safety across Modalities*, 2026. URL https://openreview.net/forum?id=1QcY6LPcdQ.

Kang Chen, Zhihao Liu, Tonghe Zhang, Zhen Guo, Si Xu, Hao Lin, Hongzhi Zang, Xiang Li, Quanlu Zhang, Zhaofei Yu, et al. π_{RL}: Online rl fine-tuning for flow-based vision-language-action models. *arXiv preprint arXiv:2510.25889*, 2025. URL https://arxiv.org/abs/2510.25889.

Emily Cheng and Carmen Amo Alonso. Liseco: Linear semantic control for language generation. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=a3o2pzZuvE.

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. doi: 10.1177/02783649241273668.

Qiao Gu, Yuanliang Ju, Shengxiang Sun, Igor Gilitschenski, Haruki Nishimura, Masha Itkina, and Florian Shkurti. Safe: Multitask failure detection for vision-language-action models. *Advances in Neural Information Processing Systems*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. URL https://arxiv.org/abs/2501.12948.

Bear Häon, Kaylene Caswell Stocking, Ian Chuang, and Claire Tomlin. Mechanistic interpretability for steering vision-language-action models. In *9th Annual Conference on Robot Learning*, 2025. URL https://openreview.net/forum?id=YvsUD8C9QS.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054.

Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 international conference on robotics and automation (ICRA)*, 2019. URL https://ieeexplore.ieee.org/document/8794127/.

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

Haozhan Li, Yuxin Zuo, Jiale Yu, Yuhao Zhang, Yang Zhaohui, Kaiyan Zhang, Xuekai Zhu, Yuchen Zhang, Tianxing Chen, Ganqu Cui, Dehui Wang, Dingxiang Luo, Yuchen Fan, Youbang Sun, Jia Zeng, Jiangmiao Pang, Shanghang Zhang, Yu Wang, Yao Mu, Bowen Zhou, and Ning Ding. SimpleVLA-RL: Scaling VLA training via reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=TQhSodCM4r.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, qiang liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=xzEtNSuDJk.

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-grpo: Training flow matching models via online rl. *Advances in Neural Information Processing Systems*, 2025a.

Jijia Liu, Feng Gao, Bingwen Wei, Xinlei Chen, Qingmin Liao, Yi Wu, Chao Yu, and Yu Wang. What can RL bring to VLA generalization? an empirical study. In *Advances in Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=qmBMPInbZC.

David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow matching policy gradients. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=eoEmoKoQpJ.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 2022.

John X Morris, Niloofar Mireshghallah, Mark Ibrahim, and Saeed Mahloujifar. Learning to reason in 13 parameters. *arXiv preprint arXiv:2602.04118*, 2026. URL https://arxiv.org/abs/2602.04118.

Sagnik Mukherjee, Lifan Yuan, Dilek Hakkani-Tür, and Hao Peng. Reinforcement learning finetunes small subnetworks in large language models. In *Advances in Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=0NdS4xCngO.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=TG8KACxEON.

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.

Yoonah Park, Haesung Pyun, and Yohan Jo. Bridging the knowledge-prediction gap in LLMs on multiple-choice questions. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=I0RgZXTO53.

John Schulman and Thinking Machines Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. URL https://thinkingmachines.ai/blog/lora/.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. Smolvla: A vision-language-action model for affordable and efficient robotics, 2025. URL https://arxiv.org/abs/2506.01844.

Shuhan Tan, Kairan Dou, Yue Zhao, and Philipp Krähenbühl. Interactive post-training for vision-language-action models. *arXiv preprint arXiv:2505.17016*, 2025. URL https://arxiv.org/abs/2505.17016.

Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024. URL https://arxiv.org/abs/2410.00425.

Andrew Wagenmaker, Yunchu Zhang, Mitsuhiko Nakamoto, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning. In *Proceedings of The 9th Conference on Robot Learning*, Proceedings of Machine Learning Research, 2025. URL https://proceedings.mlr.press/v305/wagenmaker25a.html.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. Vla knows its limits: Adaptive execution horizons for robot policies. In *European Conference on Computer Vision*, pp. 607–623. Springer, 2026a.

Qiuyue Wang, Mingsheng Li, Jian Guan, Jinhui Ye, Sicheng Xie, Yitao Liu, Junhao Chen, Zhixuan Liang, Jie Zhang, Xintong Hu, et al. Qwen-vla: Unifying vision-language-action modeling across tasks, environments, and robot embodiments. *arXiv preprint arXiv:2605.30280*, 2026b. URL https://arxiv.org/abs/2605.30280.

Yibin Wang, Zhimin Li, Yuhang Zang, Yujie Zhou, Jiazi Bu, Chunyu Wang, Qinglin Lu, Cheng Jin, and Jiaqi Wang. Pref-GRPO: Pairwise preference reward-based GRPO for stable text-to-image reinforcement learning, 2026c. URL https://openreview.net/forum?id=rUDEPUZAvL.

Wenli Xiao, Haotian Lin, Andy Peng, Haoru Xue, Tairan He, Zhengyi Luo, Yuqi Xie, Fengyuan Hu, Linxi Fan, Guanya Shi, and Yuke Zhu. Self-improving vision-language-action models with data generation via residual RL. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=eUGoqrZ6Ea.

Charles Xu, Jost Tobias Springenberg, Michael Equi, Ali Amin, Adnan Esmail, Sergey Levine, and Liyiming Ke. Rl token: Bootstrapping online rl with vision-language-action models. *arXiv preprint arXiv:2604.23073*, 2026. URL https://arxiv.org/abs/2604.23073.

Zeyue Xue, Jie Wu, Yu Gao, Fangyuan Kong, Lingting Zhu, Mengzhao Chen, Zhiheng Liu, Wei Liu, Qiushan Guo, Weilin Huang, and Ping Luo. Dancegrpo: Unleashing grpo on visual generation, 2025. URL https://arxiv.org/abs/2505.07818.

Qingyu Yin, Yulun Wu, Zhennan Shen, Sunbowen Lee, Zhilin Wang, Yanshu Li, Chak Tou Leong, Jiale Kang, and Jinjin Gu. Evaluating parameter efficient methods for RLVR. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=76CIL1O0bz.

Chao Yu, Yuanqing Wang, Zhen Guo, Hao Lin, Si Xu, Hongzhi Zang, Quanlu Zhang, Yongji Wu, Chunyang Zhu, Junhao Hu, Zixiao Huang, Mingjie Wei, Yuqing Xie, Ke Yang, Bo Dai, Zhexuan Xu, Jiakun Du, Xiangyuan Wang, Xu Fu, Letong Shi, Zhihao Liu, Kang Chen, Weilin Liu, Gang Liu, Boxun Li, Jianlei Yang, Zhi Yang, Guohao Dai, and Yu Wang. RLinf: Flexible and efficient Large-Scale reinforcement learning via Macro-to-Micro flow transformation. In *20th USENIX Symposium on Operating Systems Design and Implementation (OSDI 26)*, 2026. URL https://www.usenix.org/conference/osdi26/presentation/yu-chao.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, 2020. URL https://arxiv.org/abs/1910.10897.

Hongzhi Zang, Mingjie Wei, Si Xu, Yongji Wu, Zhen Guo, Yuanqing Wang, Hao Lin, Liangzhi Shi, Yuqing Xie, Zhexuan Xu, et al. Rlinf-vla: A unified and efficient framework for vla+ rl training. *arXiv preprint arXiv:2510.06710*, 2025. URL https://arxiv.org/abs/2510.06710.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. Reasoning models know when they're right: Probing hidden states for self-verification. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=O6I0Av7683.

Jiachen Zhang, Junnan Nie, Junyi Lao, Wei Cheng, Chenghao Liu, Jiaxin Jiang, and Songfang Huang. What frozen vlas already know about success: A probing study of value-like structure in foundation robot policies. *arXiv preprint arXiv:2605.28527*, 2026a.

Jiaying Zhang, Lei Shi, Jiguo Li, Jun Xu, Jiuchong Gao, Jinghua Hao, and Renqing He. GeoRA: Geometry-aware low-rank adaptation for RLVR. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026b. URL https://aclanthology.org/2026.acl-long.1110/.

Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Proceedings of Robotics: Science and Systems*, 2023. URL https://roboticsproceedings.org/rss19/p016.html.

Kaiwen Zheng, Huayu Chen, Haotian Ye, Haoxiang Wang, Qinsheng Zhang, Kai Jiang, Hang Su, Stefano Ermon, Jun Zhu, and Ming-Yu Liu. Diffusionnft: Online diffusion reinforcement with forward process. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=VJZ477R89F.

Tinghui Zhu, Sheng Zhang, James Y. Huang, Selena Song, Xiaofei Wen, Yuankai Li, Hoifung Poon, and Muhao Chen. Video models can reason with verifiable rewards, 2026. URL https://arxiv.org/abs/2605.15458.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R Sanketi, Grecia Salazar, Michael S Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J Joshi, Alex Irpan, brian ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *7th Annual Conference on Robot Learning*, 2023. URL https://openreview.net/forum?id=XMQgwiJ7KSX.

APPENDIX CONTENTS

A VLA Architectures and Timestep Modules **16**

A.1 $\pi_{0.5}$: Time MLP and AdaRMS 16

A.2 GR00T: AdaLayerNorm and Action–Timestep Encoding 16

A.3 SmolVLA: Action–Timestep MLP 17

B Experiment Settings **18**

B.1 Model List 18

B.2 In-house Training Setup 18

C Extended Results **20**

C.1 Threshold Ablations on Parameter Shift Analyses 20

C.2 Parameter Updates on Flow-Based VLA Models 20

C.3 Parameter Updates in Image and Video DiT Models 22

C.4 Low-Rank Parameter Replacement 23

C.5 Targeted LoRA 23

C.6 Time Conditioning Directions in AdaRMS Updates 24

C.7 Comparing Scale, Shift, and Gate 32

C.8 Probing Results 34

C.9 Steering Results 38

A VLA ARCHITECTURES AND TIMESTEP MODULES

In this section, we detail the Timestep Modules in the action experts of the flow-based VLAs $\pi_{0.5}$, GR00T N1.5/N1.6, and SmolVLA. Table 4 summarizes each model's architecture and parameter counts.

Table 4: **Architecture and size of Timestep Modules.** For GR00T, the count includes the shared Time MLP, the adaptive normalization projections, and the timestep-input blocks of the embodiment-specific action encoders. For SmolVLA, it includes only the timestep-input block of the action–timestep MLP, not the full joint MLP.

	$\pi_{0.5}$	GR00T N1.5	GR00T N1.6	SmolVLA
Expert hidden size	1,024	1,536	1,536	720
Adaptive normalization	AdaRMSNorm	AdaLayerNorm	AdaLayerNorm	–
Modulation outputs	scale, shift, gate	scale, shift	scale, shift	–
Timestep parameters (M)	118.60	158.52	234.07	0.52
Action expert parameters (M)	430.10	1,068.81	1,418.62	99.88
Timestep share of action expert (%)	27.58	14.83	16.50	0.52

Scope and counting convention. We define Timestep Modules as the parameters that directly map timestep embeddings to conditioning vectors, such as AdaRMS projections, and thus exclude parameters that do not transform the timestep embeddings, such as attention and feed-forward layers. For modules that take both timestep and action embeddings, we count only the weights corresponding to the timestep input and exclude the action-input block, the joint bias, and downstream projections of mixed features. We count shared parameters once, even if they are reused during inference.

A.1 $\pi_{0.5}$: TIME MLP AND ADARMS

The Timestep Modules of $\pi_{0.5}$ (Intelligence et al., 2025) consist of a shared Time MLP and layer-specific adaptive RMS normalization (AdaRMS) projections. The action expert has 18 transformer layers with hidden size $d = 1024$. The Time MLP maps a sinusoidal timestep embedding to a conditioning vector shared across layers:

$$c_\tau = \mathrm{SiLU}(W_2\,\mathrm{SiLU}(W_1\phi(\tau) + b_1) + b_2)\,, \tag{8}$$

where $W_1, W_2 \in \mathbb{R}^{1024\times1024}$.

The AdaRMS modules then produce three conditioning vectors: the scale, shift, and gate vectors. Notably, each transformer layer contains two different AdaRMS modules, one before the attention sublayer and the other before the feed-forward sublayer. For sublayer ℓ, with attention or feed-forward transformation F_ℓ, AdaRMS operates as follows:

$$[s_\ell; b_\ell; g_\ell] = W_\ell c_\tau + d_\ell, \tag{9}$$

$$z_\ell = (1 + s_\ell) \odot \mathrm{RMS}(h_\ell) + b_\ell, \tag{10}$$

$$h'_\ell = h_\ell + g_\ell \odot F_\ell(z_\ell). \tag{11}$$

Each AdaRMS projection has shape 3072×1024, consisting of equal-size 1024×1024 blocks for its three outputs. As described in Table 4, Timestep Modules account for 27.58% of the action expert, with AdaRMS accounting for the majority (98.23%).

A.2 GR00T: ADALAYERNORM AND ACTION–TIMESTEP ENCODING

The Timestep Modules of GR00T N1.5 and N1.6 incorporate timestep information through two mechanisms: adaptive LayerNorm and a joint action–timestep encoder. The action experts of GR00T N1.5 and N1.6 have 16 and 32 transformer layers, respectively, both with hidden size $d = 1536$.

Time MLP and adaptive LayerNorm. Similar to $\pi_{0.5}$, a shared Time MLP maps a 256-dimensional sinusoidal timestep embedding to a 1536-dimensional conditioning vector:

$$c_\tau = W_2 \,\mathrm{SiLU}(W_1 \phi(\tau) + b_1) + b_2, \tag{12}$$

where $W_1 \in \mathbb{R}^{1536\times 256}$ and $W_2 \in \mathbb{R}^{1536\times 1536}$. Each transformer block in the action expert then produces scale and shift vectors:

$$[s_\ell; b_\ell] = W_\ell \,\mathrm{SiLU}(c_\tau) + d_\ell, \tag{13}$$

$$z_\ell = (1 + s_\ell) \odot \mathrm{LN}(h_\ell) + b_\ell. \tag{14}$$

GR00T's LayerNorm, unlike $\pi_{0.5}$'s RMS normalization, subtracts the feature mean before normalization and does not produce a timestep-dependent residual gate. Each block-level projection has shape 3072×1536, consisting of equal-size 1536×1536 blocks for its two outputs, the scale and shift vectors. Together, the Time MLP and adaptive LayerNorm contain 83.02M and 158.57M parameters in N1.5 and N1.6, respectively.

Action–timestep encoder. GR00T additionally uses an action–timestep encoder to combine timestep information with noisy actions before the transformer:

$$e_a = A_1 a_\tau + q_1, \tag{15}$$

$$u_\tau = \mathrm{SiLU}(A_2[e_a; \phi_a(\tau)] + q_2)\,, \tag{16}$$

$$e_{\mathrm{action}} = A_3 u_\tau + q_3. \tag{17}$$

Importantly, these projections are embodiment-specific. As e_a and $\phi_a(\tau)$ each have dimension 1536, $A_2 \in \mathbb{R}^{1536\times 3072}$ for each embodiment. We partition it as:

$$A_2 = [A_{2,\mathrm{action}}, A_{2,\mathrm{time}}], \tag{18}$$

where each block has shape 1536×1536. We include $A_{2,\mathrm{time}}$ in the Timestep Modules, but exclude $A_{2,\mathrm{action}}$ and the subsequent joint projection A_3. The timestep-input blocks contain $32 \times 1536 \times 1536 = 75.50$M parameters across the stored embodiment-specific encoders. Overall, the Timestep Modules contain 158.52M parameters in N1.5 and 234.07M in N1.6, accounting for 14.83% and 16.50% of their complete action experts, respectively. Notably, while the adaptive-normalization vectors depend only on the timestep, the action-token embeddings depend jointly on the timestep and the noisy action.

A.3 SmolVLA: Action–Timestep MLP

SmolVLA uses a different architecture, with an action–timestep MLP rather than adaptive normalization. During inference, each noisy action is projected to an embedding, concatenated with a sinusoidal timestep embedding, and passed through a two-layer MLP:

$$e_a = W_a a_\tau + b_a, \tag{19}$$

$$e_{\mathrm{action}} = W_{\mathrm{out}} \,\mathrm{SiLU}(W_{\mathrm{in}}[e_a; \phi(\tau)] + b_{\mathrm{in}}) + b_{\mathrm{out}}. \tag{20}$$

The resulting embeddings simply serve as input tokens to the action expert, and thus SmolVLA does not generate timestep-dependent scale, shift, or gate vectors at each transformer layer. The action and timestep embeddings for SmolVLA each have dimension 720. Thus, $W_{\mathrm{in}} \in \mathbb{R}^{720\times 1440}$ and $W_{\mathrm{out}} \in \mathbb{R}^{720\times 720}$. We partition the input matrix as:

$$W_{\mathrm{in}} = [W_{\mathrm{action}}, W_{\mathrm{time}}], \tag{21}$$

and separately analyze the two 720×720 blocks. As with GR00T, we include only W_{time} in the Timestep Module. Overall, the timestep-input block contains 0.52M parameters, accounting for only 0.52% of the action expert.

B EXPERIMENT SETTINGS

This section details the experiment settings used. All experiments used 4 NVIDIA H100 80GB HBM3 GPUs paired with an Intel Xeon Platinum 8480+.

B.1 MODEL LIST

We use both publicly released and in-house-trained checkpoints from multiple flow-based VLA model families. Table 5 lists all public checkpoints and their comparison references, grouped by architecture, with their Hugging Face links. We use the official OpenPI release for the $\pi_{0.5}$ base checkpoint, distributed at `gs://openpi-assets/checkpoints/pi05_base`. All other $\pi_{0.5}$ and GR00T checkpoints are released by RLinf (Zang et al., 2025; Yu et al., 2026). We use Flow-SDE-based $\pi_{0.5}$ RL checkpoints for MetaWorld, ManiSkill, and CALVIN. All RL checkpoints are trained with PPO, except for SmolVLA, which uses GRPO (Shao et al., 2024).

Table 5: **Public checkpoints and comparison references.**

Update	Benchmark	Reference	Post-trained checkpoint
$\boldsymbol{\pi_{0.5}}$			
BC	LIBERO	OpenPI $\pi_{0.5}$ base	LIBERO few-shot SFT
BC	MetaWorld	OpenPI $\pi_{0.5}$ base	MetaWorld SFT
BC	ManiSkill	OpenPI $\pi_{0.5}$ base	ManiSkill-25Main SFT
BC	CALVIN	OpenPI $\pi_{0.5}$ base	CALVIN ABC-D SFT
BC	LIBERO	LIBERO few-shot SFT	LIBERO full-shot SFT
RL	MetaWorld	MetaWorld SFT	MetaWorld RL-FlowSDE
RL	ManiSkill	ManiSkill-25Main SFT	ManiSkill-25Main RL-FlowSDE
RL	CALVIN	CALVIN ABC-D SFT	CALVIN ABC-D RL-FlowSDE
GR00T N1.5			
RL	LIBERO-Spatial	Spatial SFT	Spatial RL, step 400
RL	LIBERO-Object	Object SFT	Object RL, step 400
RL	LIBERO-Goal	Goal SFT	Goal RL, step 500
GR00T N1.6			
RL	LIBERO-Spatial	Spatial SFT	Spatial RL, step 500
SmolVLA (community releases)			
BC	LIBERO-Spatial	SmolVLA base	Spatial SFT
BC	LIBERO-Spatial	Spatial SFT	Spatial RS-SFT
RL	LIBERO-Spatial	Spatial SFT	Spatial GRPO
RL	LIBERO-Object	Object SFT	Object GRPO, update 300

B.2 IN-HOUSE TRAINING SETUP

We also train both BC and RL policies in-house. Specifically, we train (1) $\pi_{0.5}$ RL on LIBERO-Spatial, LIBERO-Object, and LIBERO-Goal, as the official RLinf checkpoints are unavailable, (2) GR00T N1.5 and GR00T N1.6 RL on LIBERO for a few steps, as the official checkpoints have a floating-point mismatch (§ C.2), and (3) $\pi_{0.5}$ BC on LIBERO-Spatial and LIBERO for task-controlled BC training and the discrete-timestep ablation (§ 4.1). For all $\pi_{0.5}$ training, we initialize from the public LIBERO SFT policy, and for GR00T training, we initialize from the respective reference checkpoints in Table 5.

We detail our RL training settings in Table 6. Following π_{RL} (Chen et al., 2025), we use PPO as our base RL algorithm and adopt Flow-SDE for exploration. We mostly follow the recommended hyperparameters, except for $\pi_{0.5}$, where we found 240 interaction steps to be sufficient for LIBERO-Object and $n = 5$ denoising steps to work better for LIBERO-Spatial. We detail our BC hyperparameter settings in Table 7.

Table 6: **In-house RL hyperparameters on LIBERO.**

	$\pi_{0.5}$			GR00T N1.5			GR00T N1.6
Hyperparameter	**Spatial**	**Object**	**Goal**	**Spatial**	**Object**	**Goal**	**Spatial**
Optimization							
Max steps	150	120	150	25	5	5	5
Global batch size	2048	2048	2048	1024	768	1024	720
Update epochs	1	1	4	4	4	4	4
Actor learning rate	5×10^{-6}	5×10^{-6}	5×10^{-6}	5×10^{-6}	5×10^{-6}	5×10^{-6}	5×10^{-6}
Critic learning rate	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}
PPO							
Discount factor γ	0.99	0.99	0.99	0.99	0.99	0.99	0.99
GAE λ	0.95	0.95	0.95	0.95	0.95	0.95	0.95
PPO clip ratio ϵ	0.2	0.2	0.2	0.2	0.2	0.2	0.2
KL coefficient	0	0	0	0	0	0	0.01
Environment reward scale	1	1	1	1	1	1	5
Rollout							
Interaction steps	240	240	320	240	240	240	240
Parallel environments	64	64	64	64	48	64	48
Rollout epochs	8	8	8	8	8	8	8
Action horizon H	10	10	10	16	16	16	50
Replan horizon H'	5	5	5	5	5	5	16
Flow Policy							
Denoising steps	5	5	5	4	4	4	4
Flow-SDE noise σ	0.5	0.3	0.3	0.5	0.5	0.5	0.5

Table 7: **In-house BC hyperparameters for $\pi_{0.5}$.**

Hyperparameter	**LIBERO**
Optimizer steps	1800
Global batch size	2048
Learning rate	5×10^{-6}
Weight decay	0.01
Gradient clipping norm	1.0
Action horizon H	10
Timestep sampling	Continuous beta

C Extended Results

This section details the additional experiment results for each sections.

C.1 Threshold Ablations on Parameter Shift Analyses

We examine whether our findings depend on the thresholds used to measure update density and effective rank. For *density threshold ablations*, we vary the absolute-change threshold from 10^{-7} to 10^{-3} using LIBERO-Spatial BC and RL checkpoints initialized from the same SFT policy and trained for 600 optimizer updates. For *rank threshold ablations*, we vary the explained-energy threshold over 90%, 95%, and 99%, using the released BC and RL checkpoint comparisons on MetaWorld, ManiSkill, and CALVIN from the main analysis.

We find that, similar to Figure 2, the Timestep Modules are updated much more densely than attention and MLP modules under both BC and RL across thresholds from 10^{-7} to 10^{-4} (Figure 5, top). At 10^{-3}, however, nearly all RL updates fall below the threshold.

We next examine effective update rank and find that the BC–RL difference in the Timestep Modules persists across all tested energy thresholds (Figure 5, bottom). Even at 99%, AdaRMS updates require only 8.2–9.3% of the available rank under RL, compared with 35.3–46.0% under BC. In contrast, attention and MLP updates have similarly high ranks under both objectives.

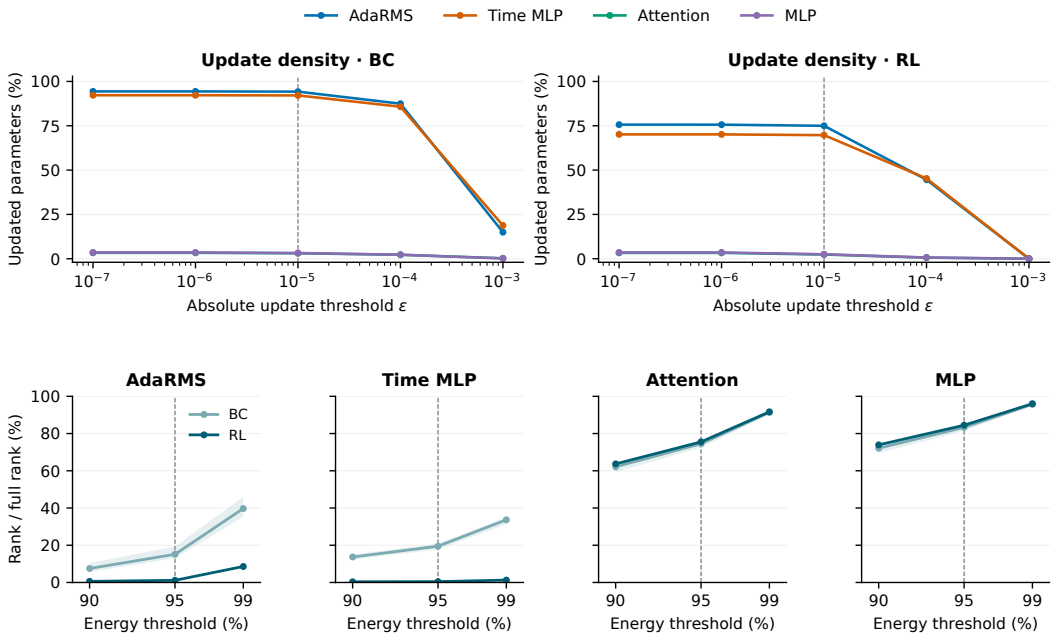


Figure 5: **Sensitivity to measurement thresholds.** (Top) Update density across absolute-change thresholds. (Bottom) Effective update rank across explained-energy thresholds.

C.2 Parameter Updates on Flow-Based VLA Models

We extend our analysis to GR00T N1.5, GR00T N1.6, and SmolVLA (Bjorck et al., 2025; Shukor et al., 2025) on LIBERO, under the same settings as the analyses in § 4.1. For *GR00T*, we analyze N1.5 RL checkpoints on Spatial, Object, and Goal, and an N1.6 checkpoint on Spatial. For *SmolVLA*, we analyze Spatial BC and RL checkpoints and an Object RL checkpoint. We separate the timestep-input block of the action–timestep MLP from its output matrix.

We find that GR00T RL updates have low effective ranks in AdaNorm and Time MLP, requiring only 3–4 directions per matrix, while attention and MLP updates require 9.8–21.6% of the available rank (Figure 6). All four components show substantial update densities.

We find a similar pattern in SmolVLA's timestep-input block: RL updates require only 7–8 directions, compared with 156 under BC (Figure 7). The output matrix requires more directions than the timestep-input block, while attention and MLP updates remain high-rank.

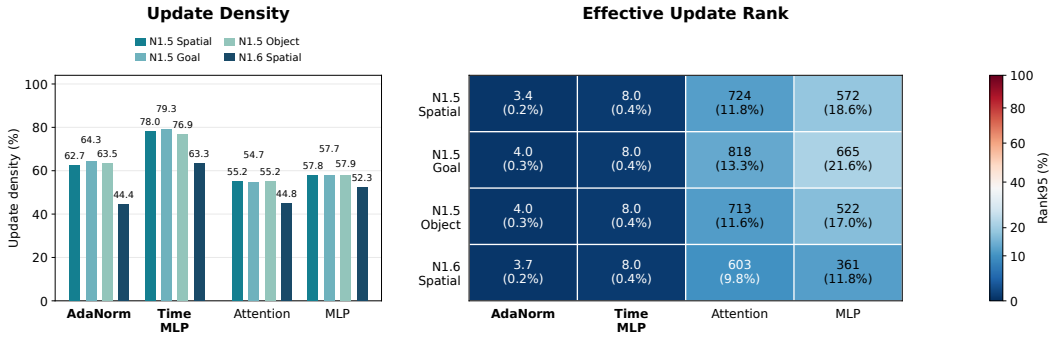


Figure 6: **GR00T RL updates on LIBERO.** (Left) Update density. (Right) Effective update rank. Parentheses indicate percentages of maximum rank.

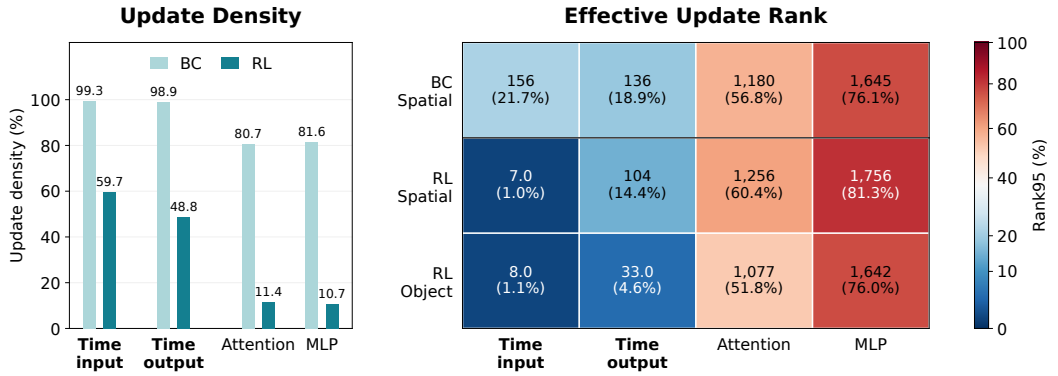


Figure 7: **SmolVLA updates on LIBERO.** (Left) Spatial BC and RL update density. (Right) Effective update rank, including Object RL. Parentheses indicate percentages of maximum rank.

C.3 Parameter Updates in Image and Video DiT Models

We extend our parameter analysis to publicly released RL-trained image and video models to examine whether the low-rank updates observed in VLA Timestep Modules also appear in other diffusion- or flow-based models. We analyze FLUX with DanceGRPO (Xue et al., 2025) and Pref-GRPO (Wang et al., 2026c), SeFi-Image with DiffusionNFT (Zheng et al., 2026), and Wan2.2 with VideoRLVR (Zhu et al., 2026) (Table 8). We group the analyzed matrices into similar timestep-related modules, such as AdaNorm and Time MLP, along with attention and MLP modules, with AdaNorm covering architecture-specific modulation projections.

We find that attention and MLP account for approximately 90–99% of the measured update energy (Figure 8, left). Effective ranks vary across models, with AdaNorm updates being relatively high-rank in SeFi-Image and VideoRLVR but lower-rank in DanceGRPO and Pref-GRPO. Time MLP updates also show substantial variation, including low-rank updates in image models (Figure 8, right).

Table 8: **Reference and RL checkpoint pairs for image and video parameter analyses.**

Model	Reference	RL checkpoint
DanceGRPO	FLUX.1-dev	xzyhku/flux_hpsv2.1_dancegrpo, checkpoint-300-0
Pref-GRPO	FLUX.1-dev	CodeGoat24/FLUX.1-dev-PrefGRPO
SeFi-Image	SeFi-Image-5B-Base-diffusers	SeFi-Image-5B-RL-diffusers
VideoRLVR	DarthZhu/VideoRLVR-Wan2.2-Base	DarthZhu/VideoRLVR-Wan2.2

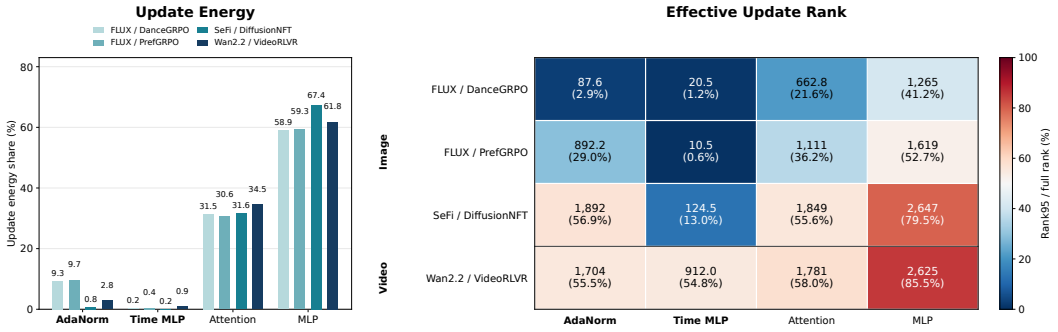


Figure 8: **RL updates in image and video models.** (Left) Update-energy share. (Right) Mean per-matrix Rank95 and normalized rank.

C.4 Low-Rank Parameter Replacement

We extend § 3.2 to test whether low-rank updates carry the RL gains in the Timestep Modules. For each weight matrix in the Timestep Modules, we compute the SVD of its RL update and reconstruct the update using either only the top four singular directions (*Top Singular Directions only*) or the remaining directions (*w/o Top Singular Directions*), with bias parameters unchanged. As shown in Table 9, the top four directions alone retain nearly all of the performance of *Timestep Modules only* for $\pi_{0.5}$, whereas removing them drops performance below the base policy on five of six benchmarks, showing that the RL gains are largely captured by the low-rank directions. For GR00T (Table 10), although the checkpoints contain high-rank noise from low-precision training, the top directions still carry most of the gain.

Table 9: **Top singular direction parameter replacement for $\pi_{0.5}$.** Best results are **bolded**.

	$\pi_{0.5}$					
Condition	Spatial	Object	Goal	ManiSkill	MetaWorld	CALVIN
Base	85.0	95.4	82.8	42.2	42.8	61.8
Timestep Modules only	**94.8**	**99.0**	**91.4**	87.5	**69.6**	**87.0**
w/o Top Singular Directions	79.6	51.8	78.4	53.8	22.6	48.7
w/ Top Singular Directions Only	90.4	96.6	89.4	**88.4**	66.6	86.7

Table 10: **Top singular direction parameter replacement for GR00T.** Best results are **bolded**.

	GR00T N1.5			GR00T N1.6
Condition	Spatial	Object	Goal	Spatial
Base	49.4	62.4	51.4	75.6
Timestep Modules only	**52.6**	**81.6**	49.8	**84.8**
w/o Top Singular Directions	51.0	65.2	46.2	81.2
w/ Top Singular Directions Only	51.8	81.2	**51.8**	84.0

C.5 Targeted LoRA

Building on Takeaway 2, we investigate applying LoRA (Hu et al., 2022) only to the Timestep Modules, especially since RLinf's default LoRA target omits them (Figure 9). We compare LoRA on the Timestep Modules against the base settings in Table 11, where targeting the Timestep Modules generally improves performance earlier in training and achieves higher peak and final success.

```
target_modules = [
    "proj", "qkv", "fc1", "fc2", "q", "kv",
    "fc3", "out_proj",
    "q_proj", "k_proj", "v_proj", "o_proj",
    "gate_proj", "up_proj", "down_proj", "lm_head",
]
# Excludes Time MLP and AdaRMS projections.
```

Figure 9: **RLinf's default LoRA targets omit Timestep Modules.** The $\pi_{0.5}$ Time MLP and AdaRMS projections are absent from RLInf Codebase.

Table 11: **Targeted LoRA on LIBERO-Spatial with $\pi_{0.5}$.** Success rates (%) over training steps with LoRA, rank=32. The higher success rate at each step is **bolded**.

LoRA target	**0**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Time MLP + AdaRMS	85.0	**85.6**	**90.0**	**91.4**	**91.8**	**92.0**	90.8	**92.0**	**94.2**	**93.0**	**93.0**
Attn + FFN	85.0	81.0	83.4	87.0	89.4	89.0	**91.6**	91.4	92.0	92.0	91.4

C.6 Time Conditioning Directions in AdaRMS Updates

We extend the timestep analysis to all 18 action-expert layers, showing the attention and MLP AdaRMS sites separately. For each joint scale–shift–gate update, we plot the top three input singular directions using $|\cos(v_i, c_\tau)|$, normalized by each curve's maximum.

Figures 10–13 compare BC and RL on MetaWorld and CALVIN. Figures 14 and 15 compare LIBERO-Spatial RL policies trained with three and five denoising steps. Figures 16 and 17 show LIBERO-40 BC trained for 30,000 optimizer updates with $\tau \in \{0, 0.2, 0.4, 0.6, 0.8\}$.

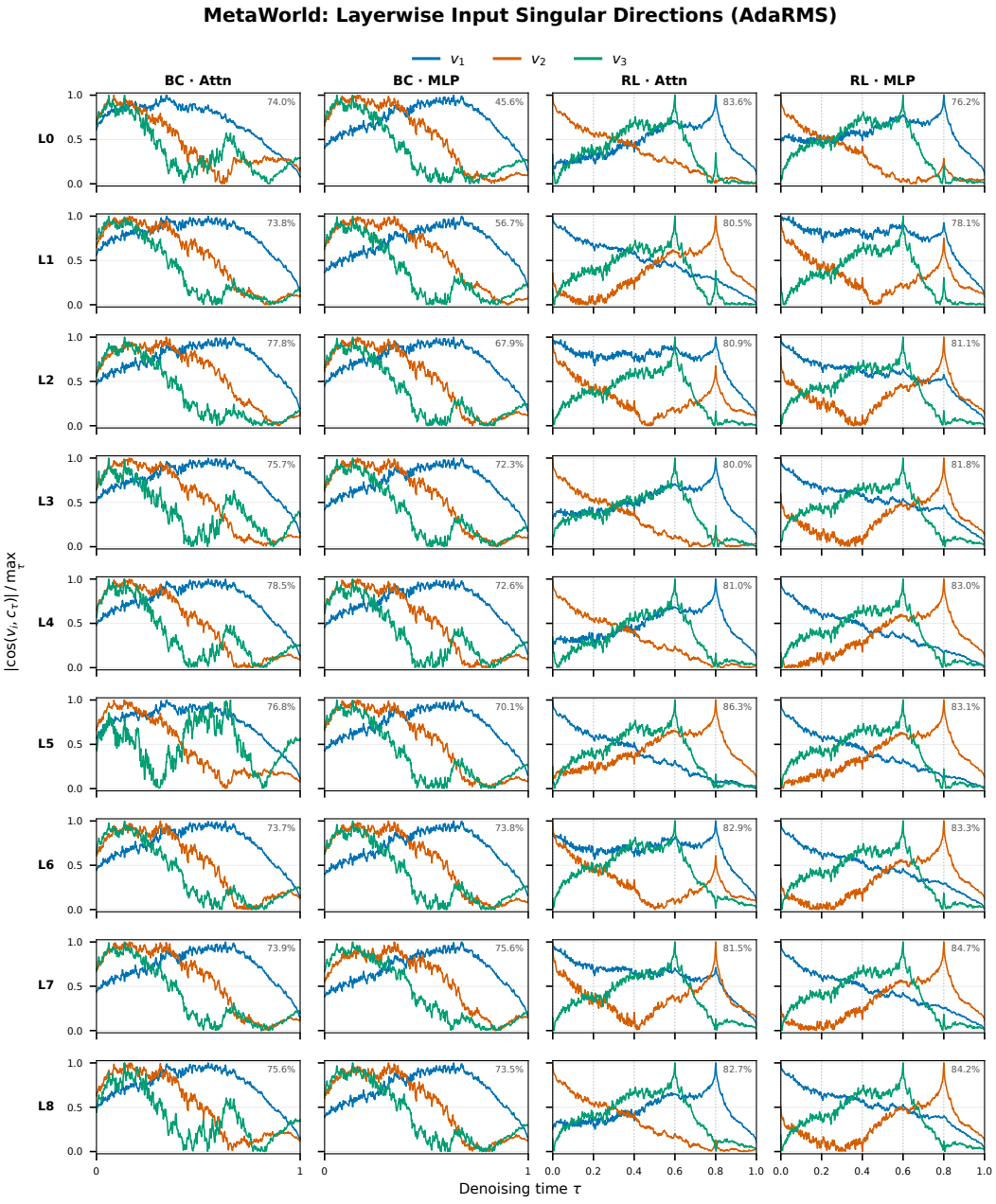


Figure 10: **Layerwise BC and RL AdaRMS input directions on MetaWorld** (Layers 0–8).

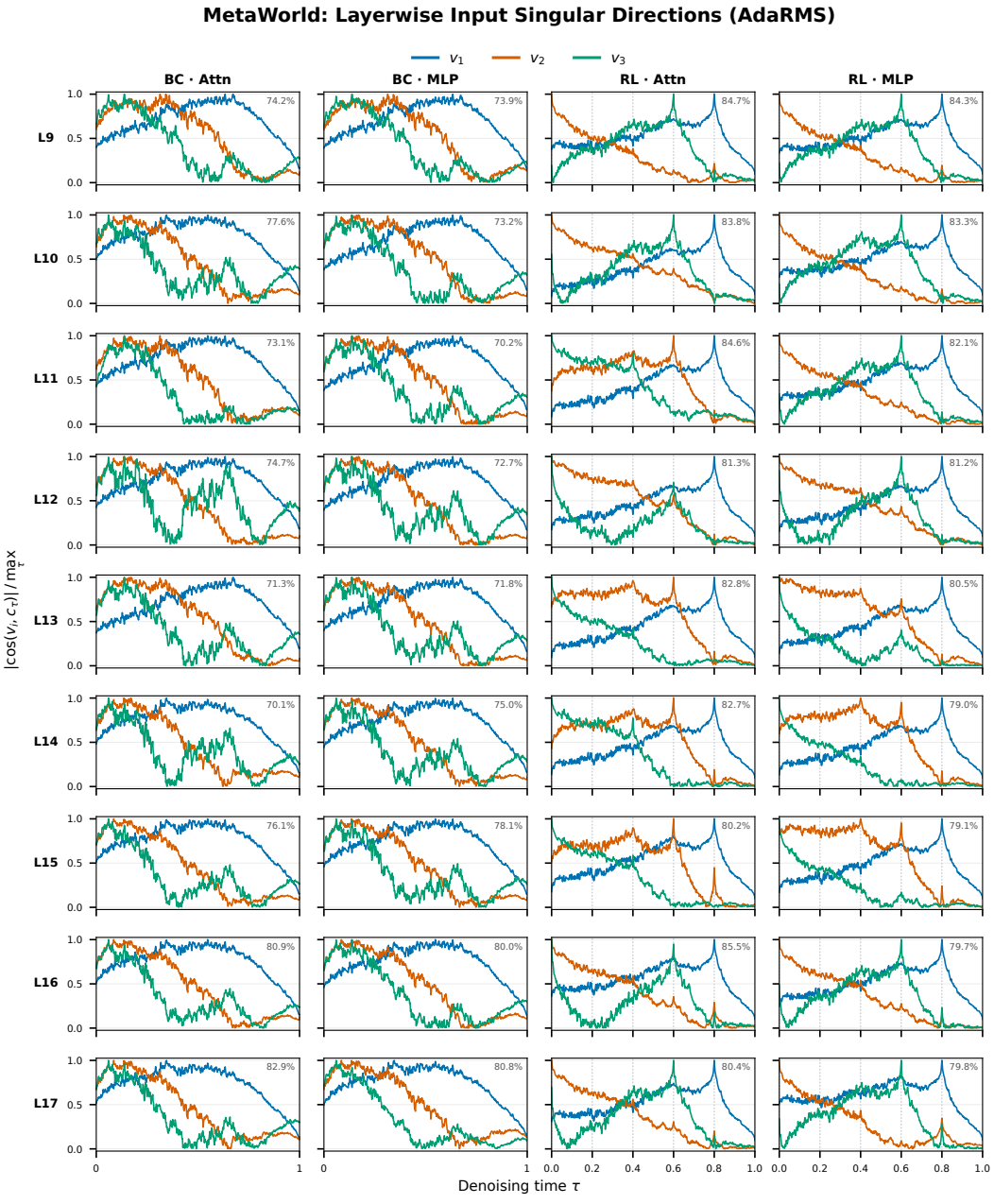


Figure 11: **Layerwise BC and RL AdaRMS input directions on MetaWorld** (Layers 9–17).

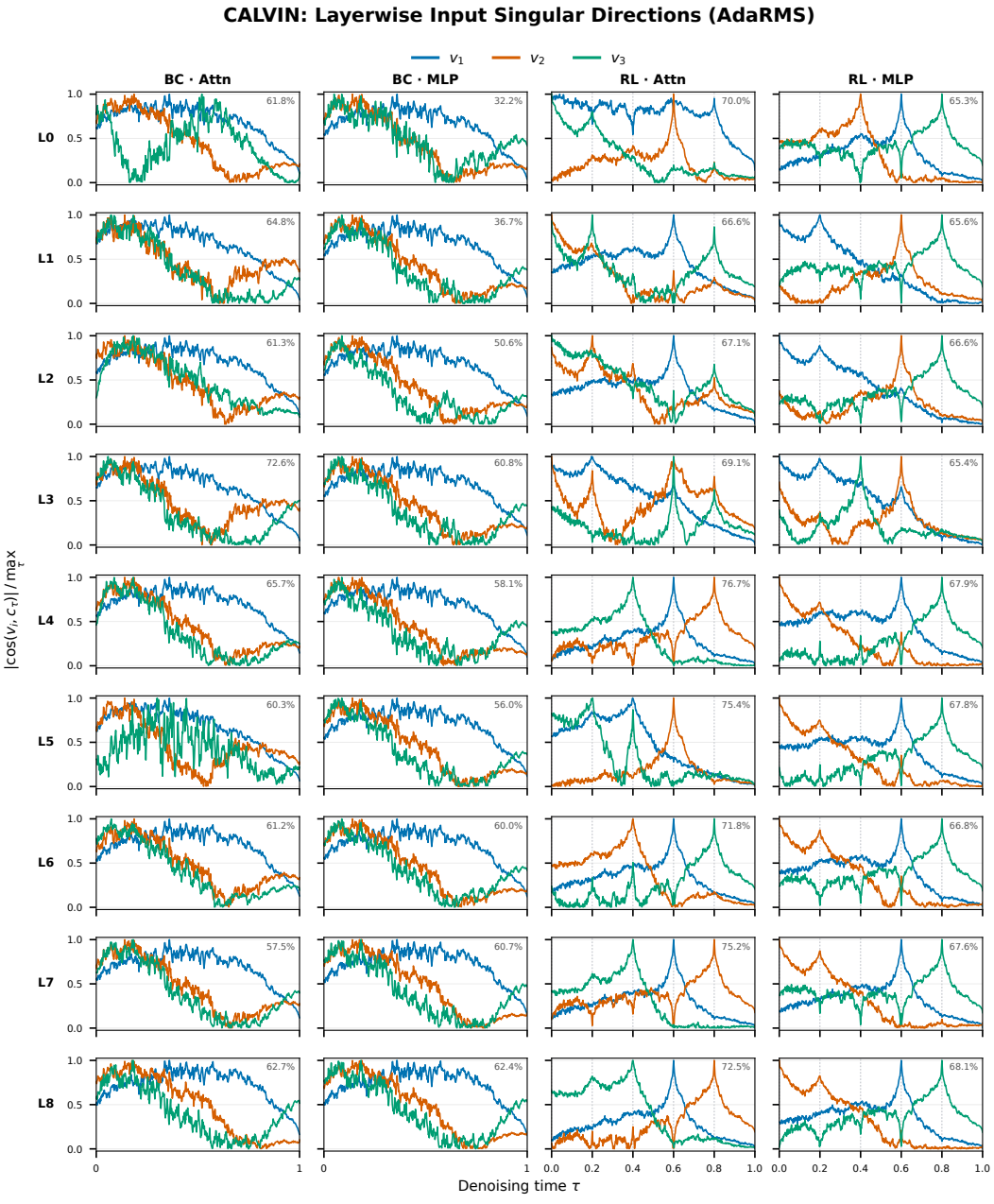


Figure 12: **Layerwise BC and RL AdaRMS input directions on CALVIN** (Layers 0–8).

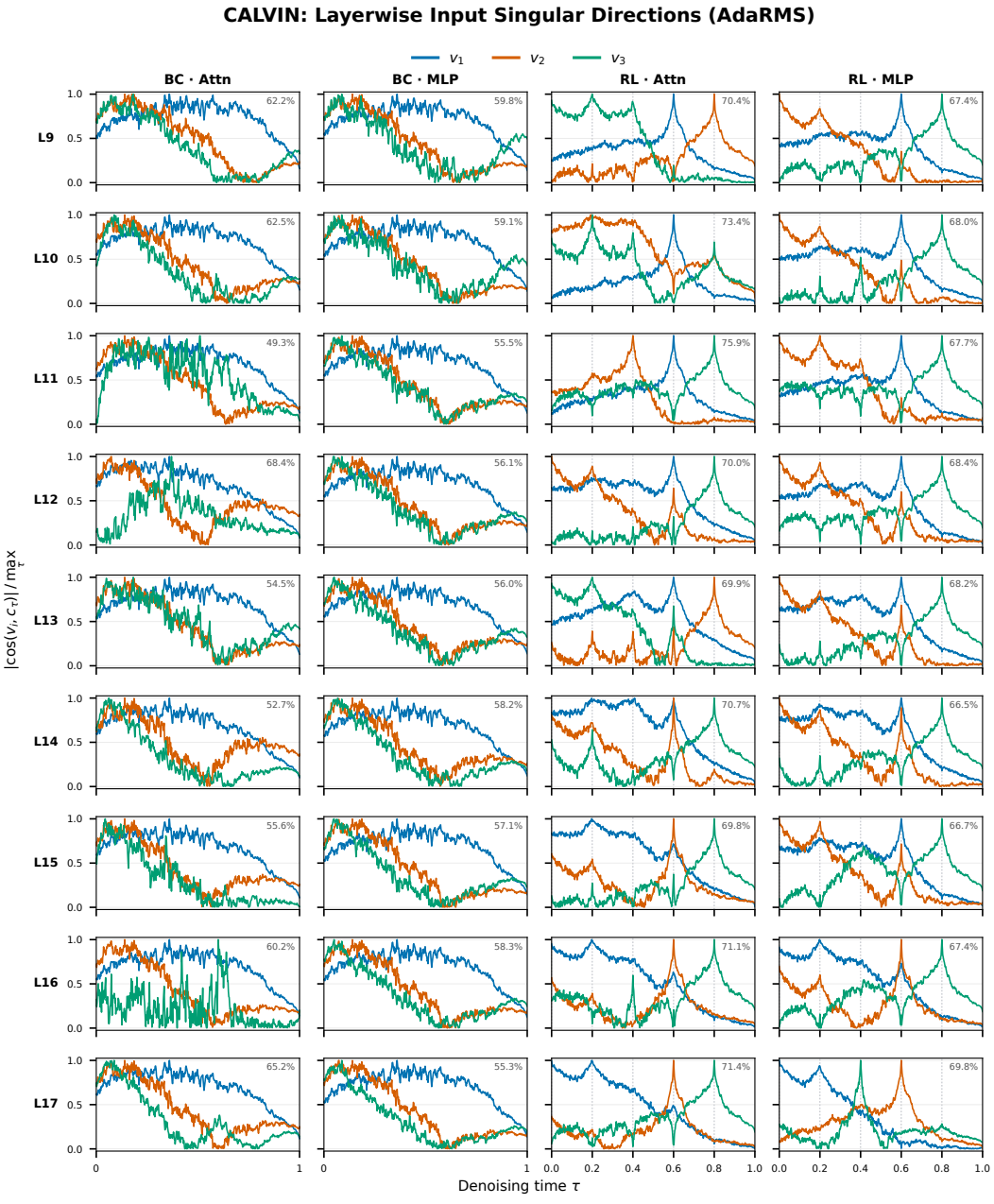


Figure 13: **Layerwise BC and RL AdaRMS input directions on CALVIN** (Layers 9–17).

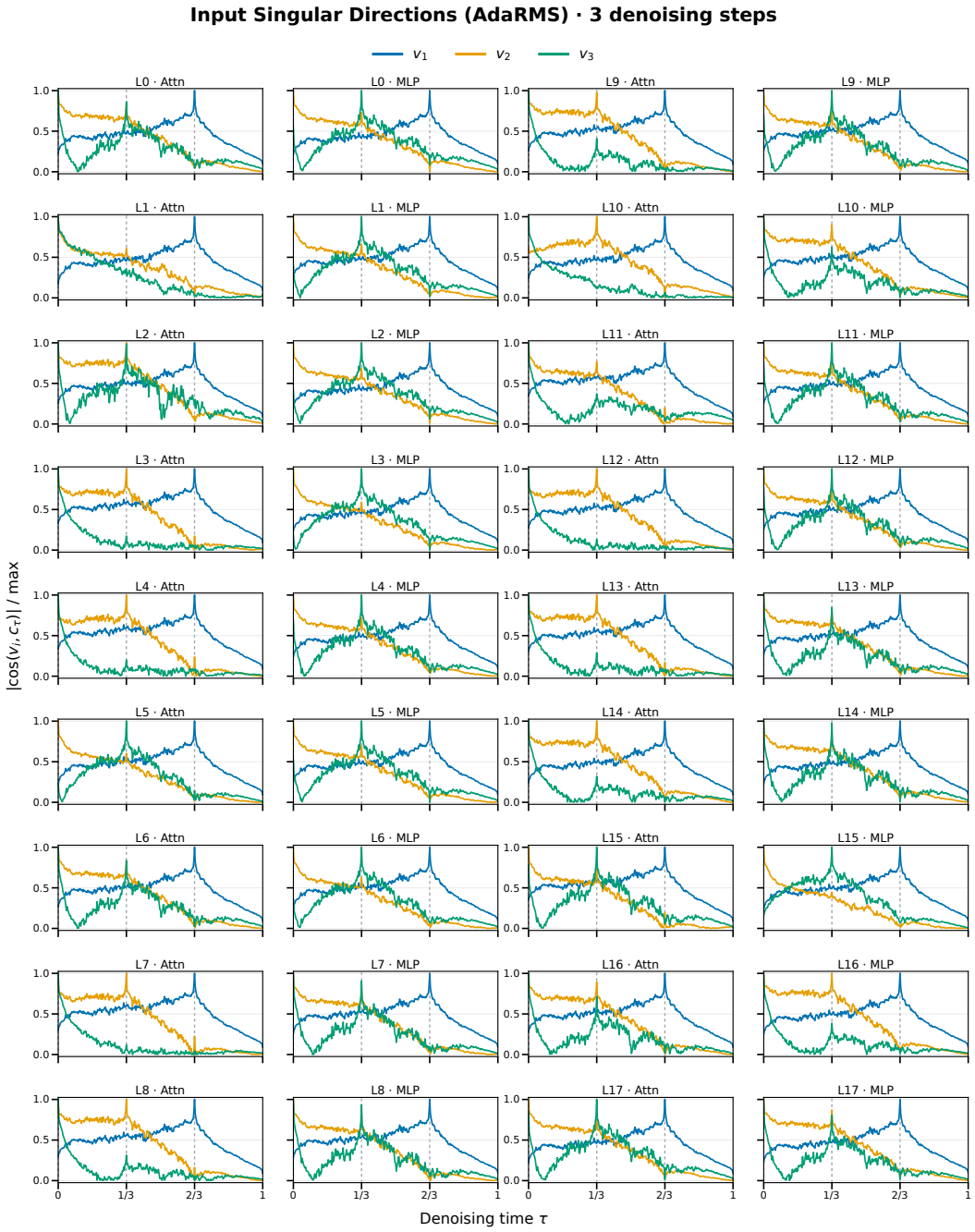


Figure 14: **Layerwise AdaRMS input directions for LIBERO-Spatial RL with three denoising steps.**

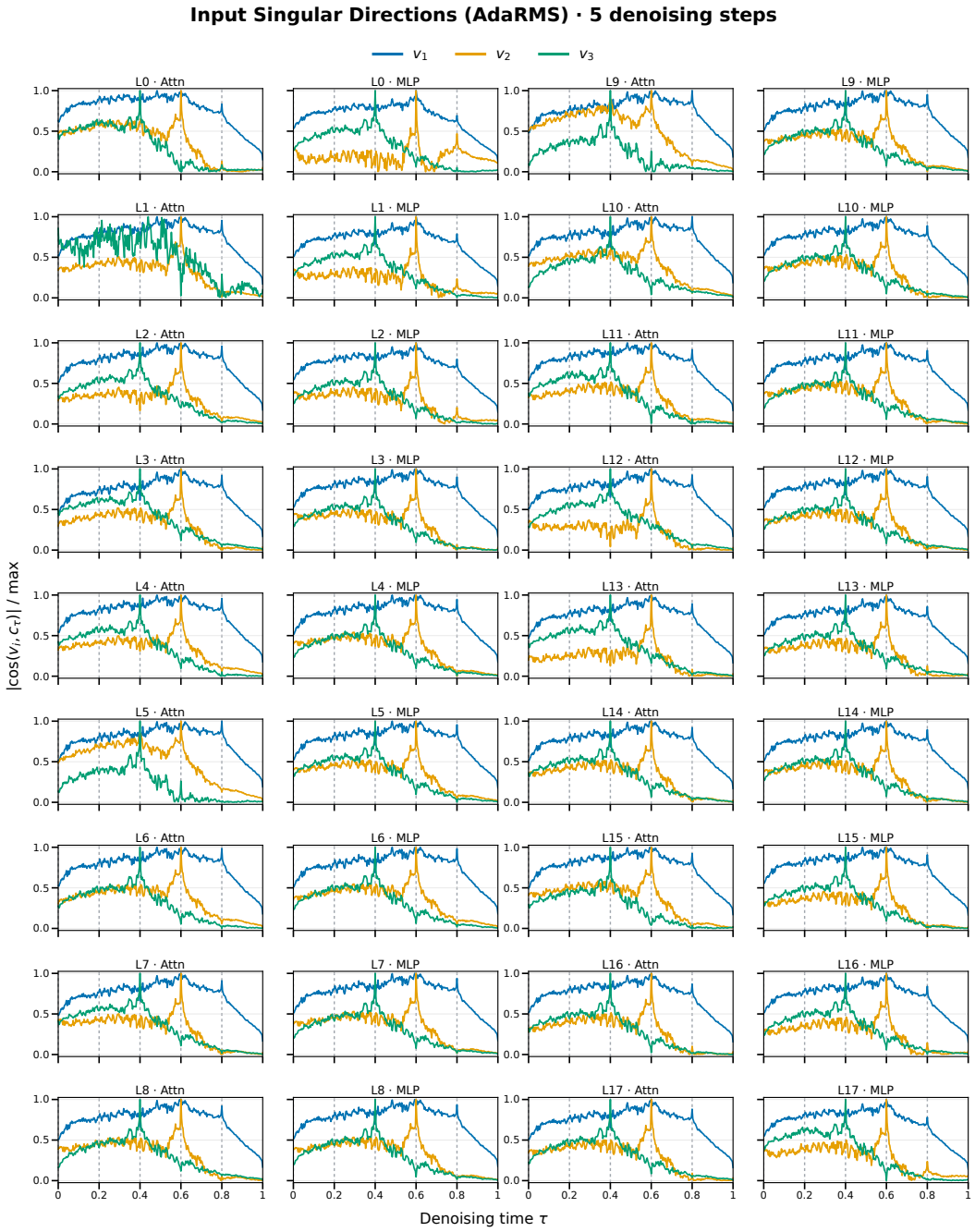


Figure 15: **Layerwise AdaRMS input directions for LIBERO-Spatial with RL five denoising steps.**

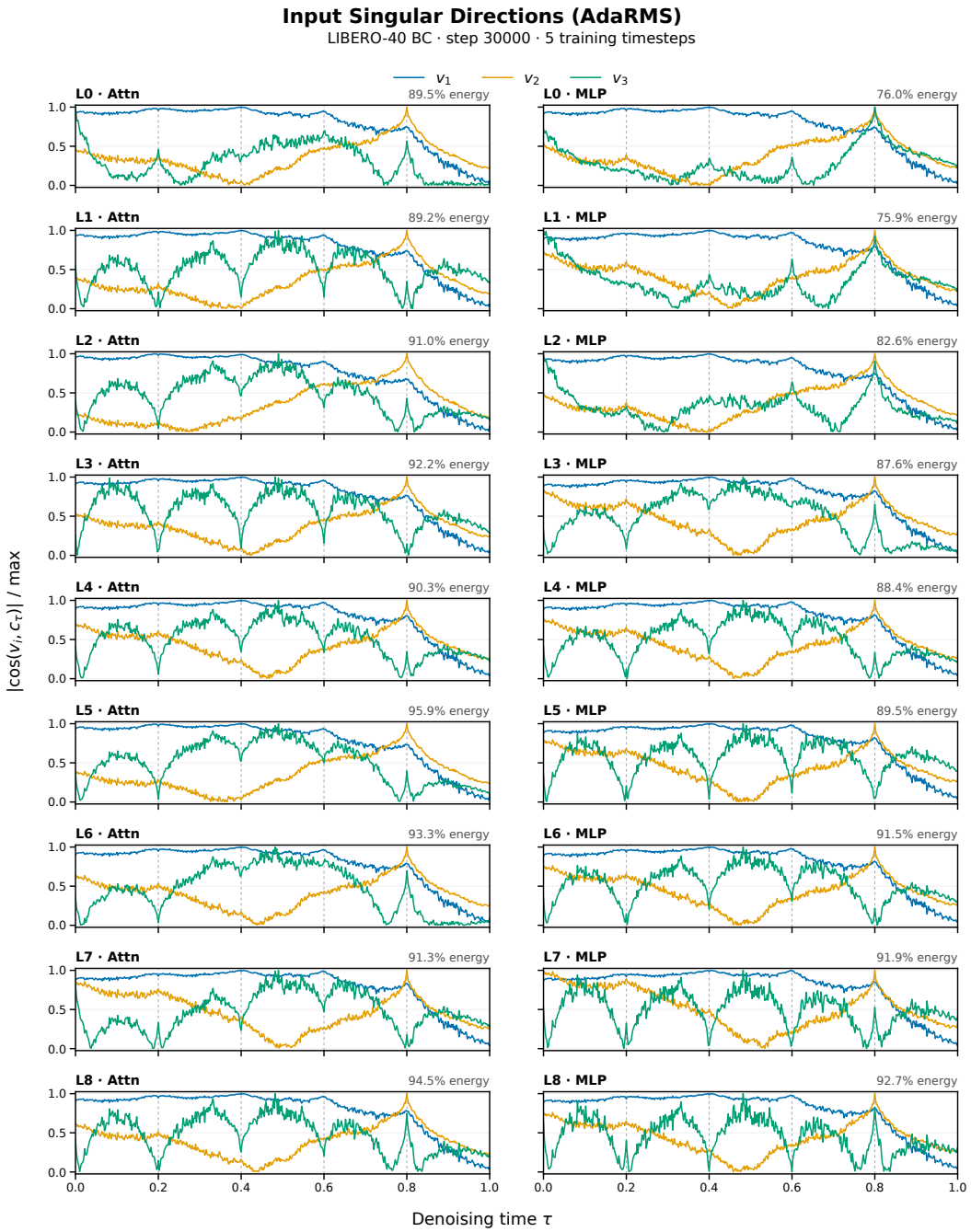


Figure 16: **Layerwise AdaRMS input directions for BC trained with five discrete timesteps** (Layers 0–8).

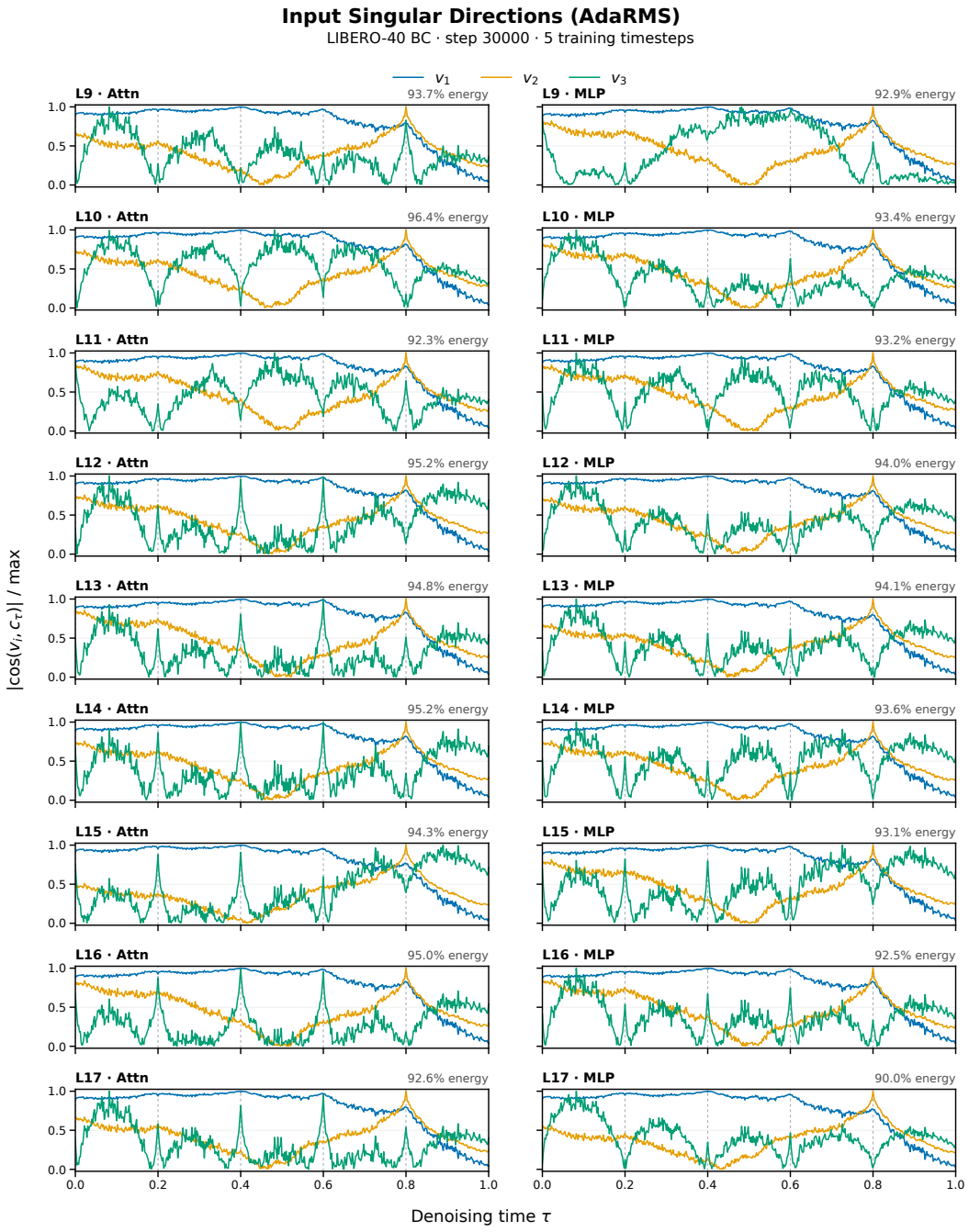


Figure 17: **Layerwise AdaRMS input directions for BC trained with five discrete timesteps** (Layers 9–17).

C.7 Comparing Scale, Shift, and Gate

Figures 18 and 19 compare RL-induced directional changes in scale, shift, and gate across $\pi_{0.5}$ and GR00T. The shift vector changes more distinctly, motivating our focus on it.

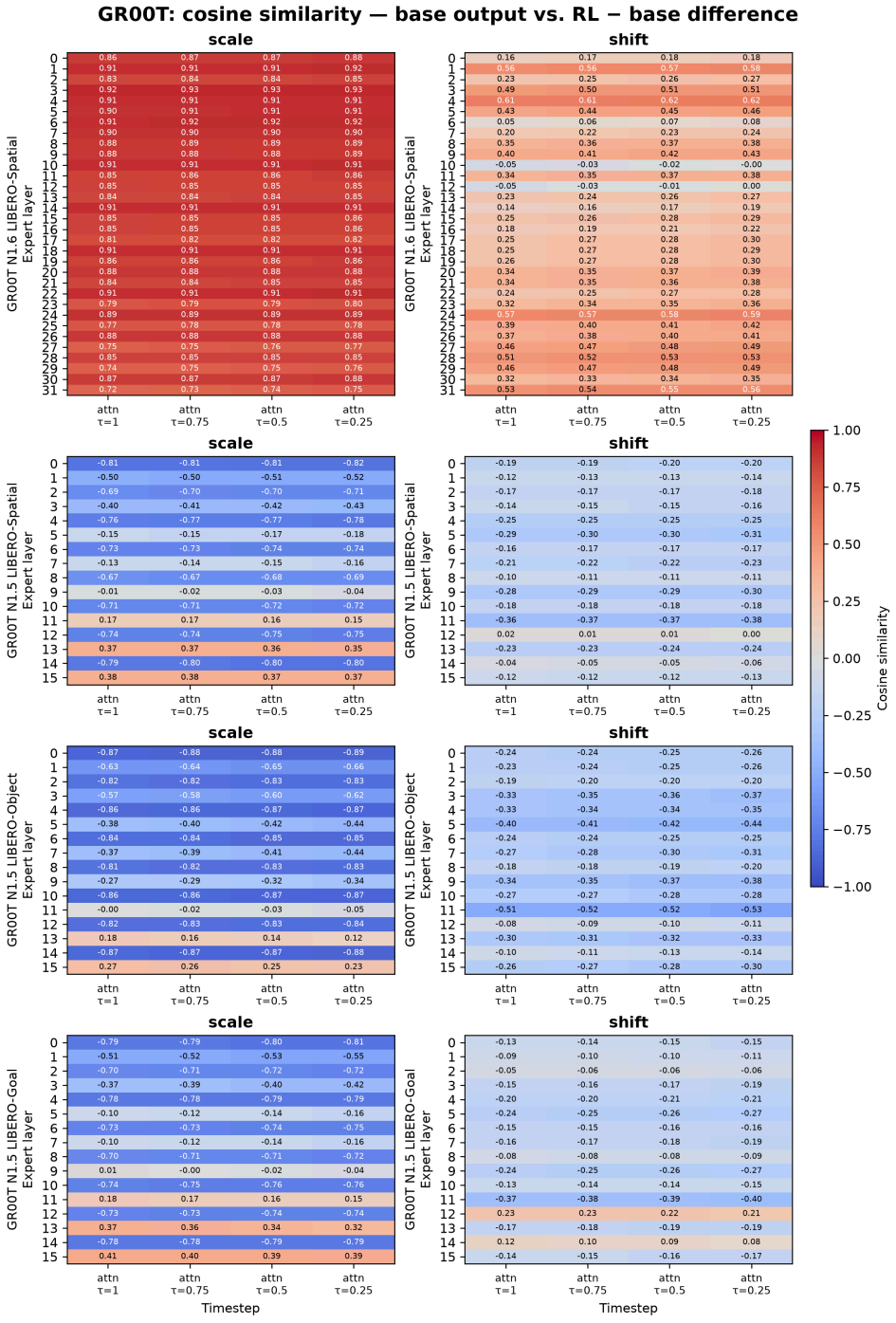


Figure 18: **Position-wise cosine similarity between each base output and its RL-induced change in GR00T checkpoints.**

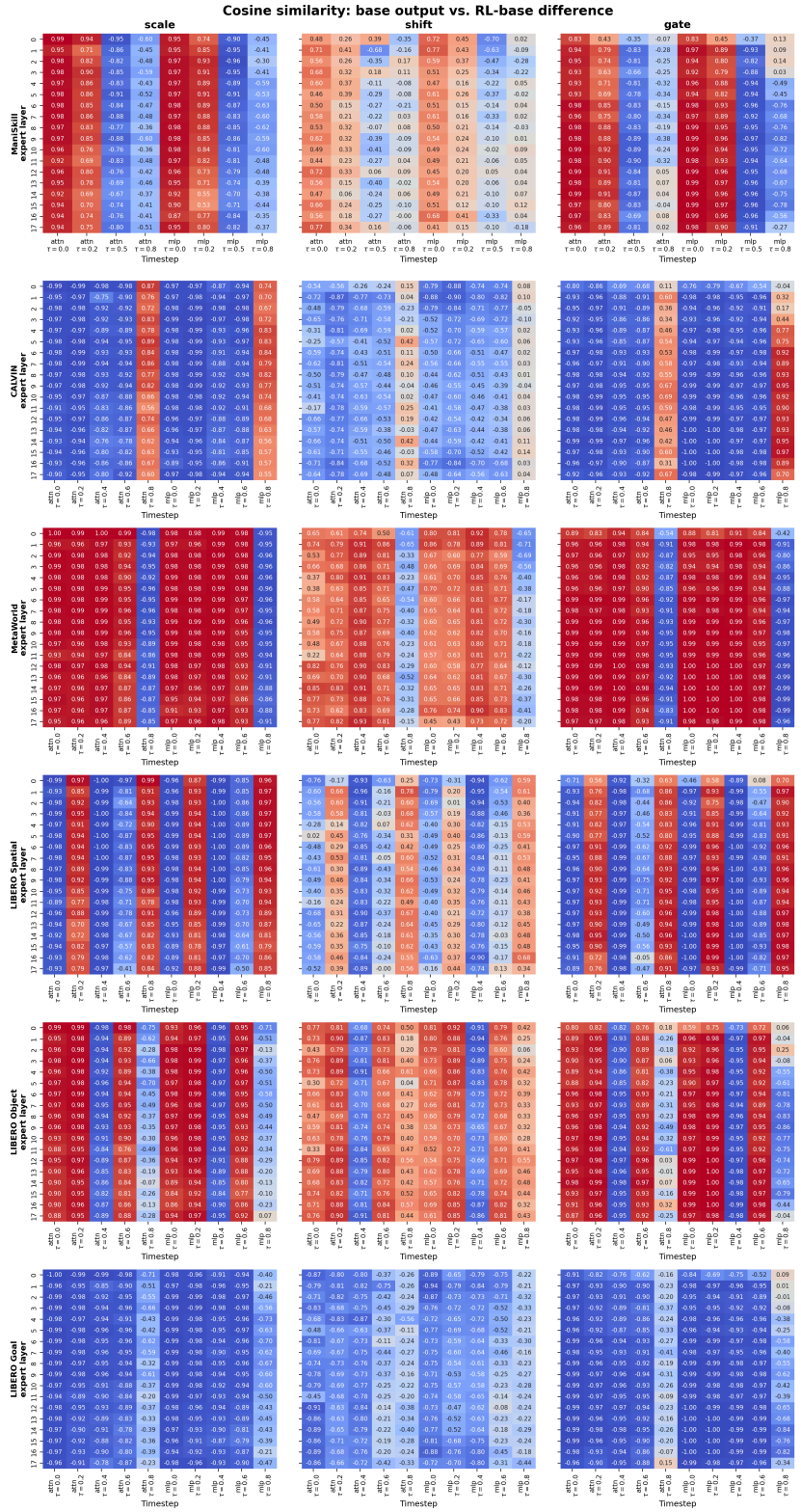


Figure 19: **Position-wise cosine similarity between each base output and its RL-induced change in $\pi_{0.5}$ checkpoints.**

C.8 PROBING RESULTS

We provide additional results on how shift vectors encode task outcomes. Table 12 shows that even a single sublayer–timestep direction can be highly predictive, reaching ROC-AUCs of 96.9–99.0 on LIBERO-Spatial, LIBERO-Object, and ManiSkill. Across individual positions, some directions show larger projections for successful episodes, while others show the opposite. These two cases appear in similar proportions for both base and RL policies (Table 13). Figures 20 and 21 visualize this structure across sublayer–timestep positions. This suggests that RL may modulate task-relevant representations by increasing or decreasing activation along different shift directions across sublayers and timesteps, with their combined effects shaping successful policy behavior.

Table 12: **Extended linear probing results with shift vectors in $\pi_{0.5}$.** F1 and ROC-AUC (%) for probes applied to the base and RL policies. We use per-benchmark 70:30 train–test splits. Logistic denotes our ℓ_2-regularized logistic regression probes. Single-Position uses a single sublayer–timestep shift vector, with the position selected by the best F1 on the training split. Random denotes probes trained on randomly shuffled outcome labels.

		Base Policy		**RL Policy**	
Benchmark	Method	F1	AUC	F1	AUC
LIBERO Spatial	Logistic	**97.1**	**99.6**	**96.3**	**96.6**
	Single-Position	93.6	99.0	85.7	78.0
	Random	58.0 ± 20.5	46.5 ± 4.0	75.2 ± 13.6	51.5 ± 3.9
LIBERO Object	Logistic	**98.2**	**98.6**	**97.3**	**97.4**
	Single-Position	97.8	98.2	**97.3**	94.7
	Random	76.3 ± 10.2	53.4 ± 6.3	86.2 ± 15.8	47.2 ± 13.3
LIBERO Goal	Logistic	**77.5**	**98.9**	**74.9**	**68.0**
	Single-Position	70.5	51.2	72.5	60.0
	Random	47.6 ± 37.2	49.0 ± 3.5	65.4 ± 28.9	52.6 ± 6.4
ManiSkill	Logistic	**95.8**	**99.6**	**98.3**	**98.3**
	Single-Position	90.7	96.9	95.6	92.0
	Random	42.9 ± 8.9	51.8 ± 7.0	87.1 ± 3.4	46.5 ± 9.3
MetaWorld	Logistic	**58.3**	**74.2**	81.1	**90.1**
	Single-Position	53.1	50.6	**82.4**	72.0
	Random	43.4 ± 7.8	49.4 ± 4.8	64.8 ± 6.6	49.4 ± 6.1

Table 13: **Probe direction at single positions in $\pi_{0.5}$.** Fraction of single positions where the original or reverse probe direction is predictive (%) across benchmarks and policies. Original indicates that larger projections are associated with success, whereas Reverse indicates the opposite.

Model	Benchmark	Original (+)	Reverse (−)
Base	LIBERO Spatial	58.4	41.6
	LIBERO Object	62.7	37.3
	LIBERO Goal	43.2	56.8
	ManiSkill	44.6	55.4
	MetaWorld	52.4	47.6
	Average	**52.6**	**47.4**
RL	LIBERO Spatial	58.9	41.1
	LIBERO Object	55.1	44.9
	LIBERO Goal	50.8	49.2
	ManiSkill	36.5	63.5
	MetaWorld	37.8	62.2
	Average	**48.3**	**51.7**

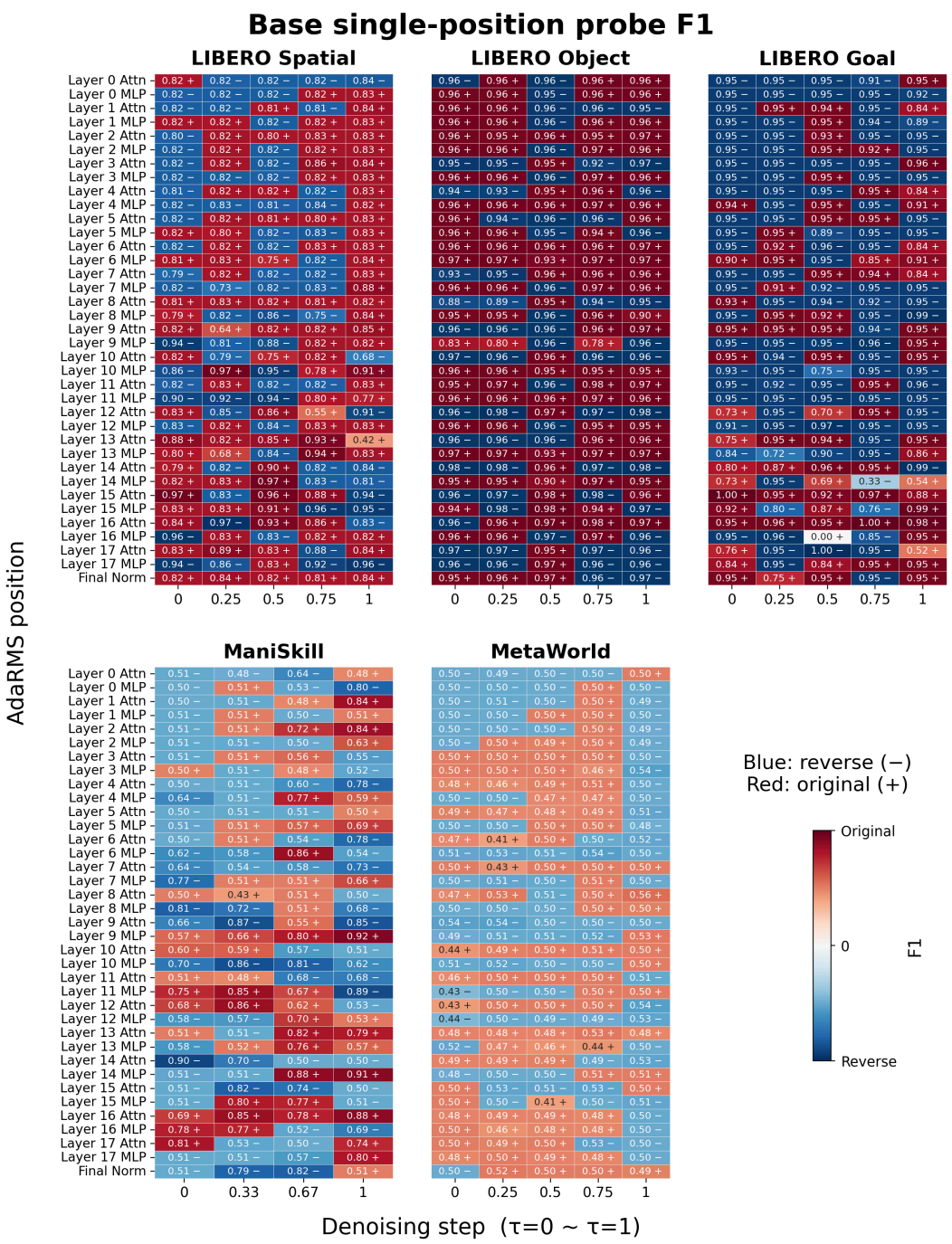


Figure 20: **Single-position probing results in $\pi_{0.5}$ base policies.** Each sublayer–timestep direction is used individually as a probe. Red indicates that larger projections are associated with success (Original), while blue indicates the opposite (Reverse). Values denote F1 scores on the test split.

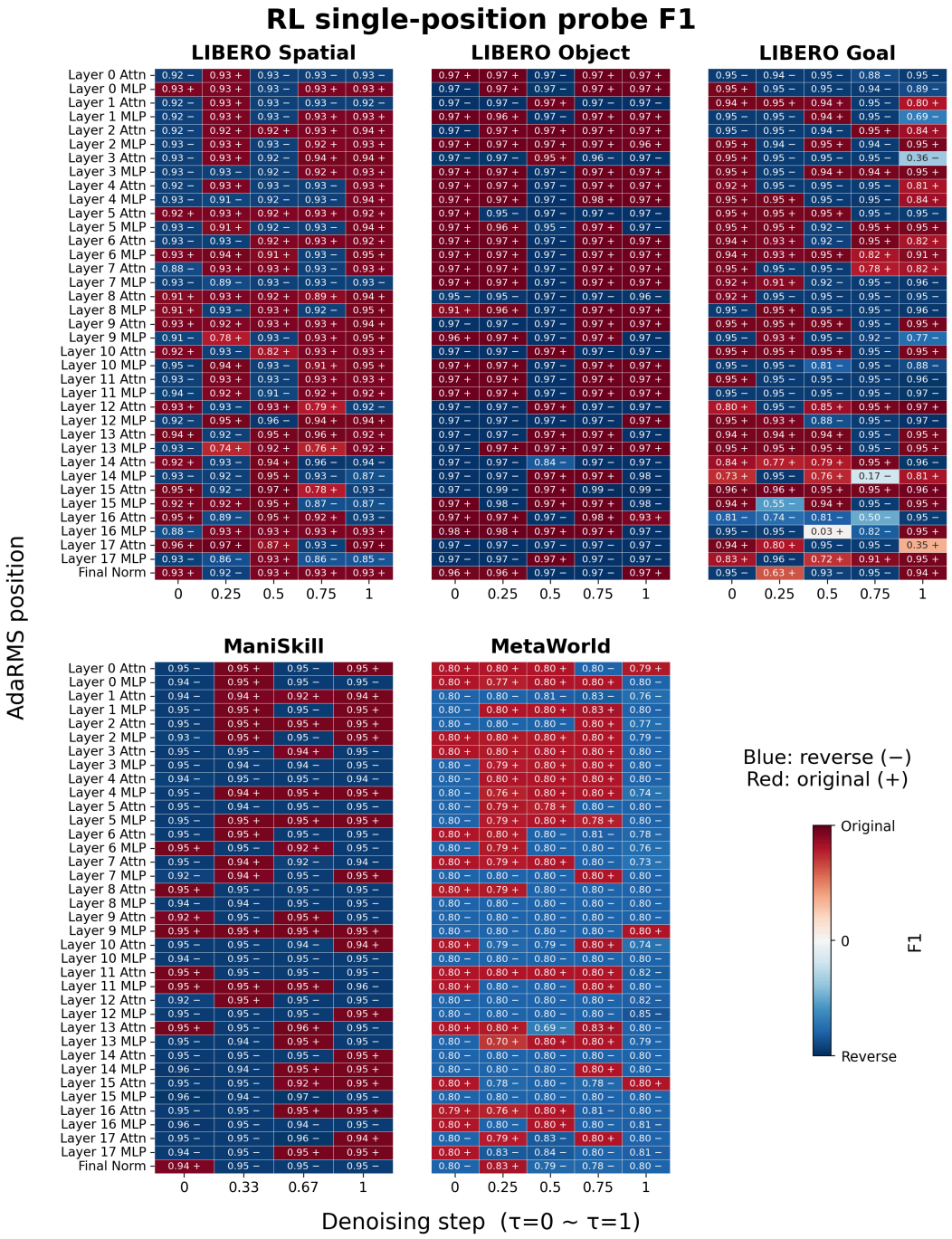


Figure 21: **Single-position probing results in $\pi_{0.5}$ RL policies.** Each sublayer–timestep direction is used individually as a probe. Red indicates that larger projections are associated with success (Original), while blue indicates the opposite (Reverse). Values denote F1 scores on the test split.

We further show the episode-level trajectories of our logistic regression probe in Figure 22. These results show that task-outcome signals arise before the end of an episode, as predicted success begins to separate between successful and failed trajectories early in execution.

For the logistic regression probes, we sweep the ℓ_2-regularization hyperparameter over $\{0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10, 30, 100\}$ and select the best value via cross-validation on the training split.

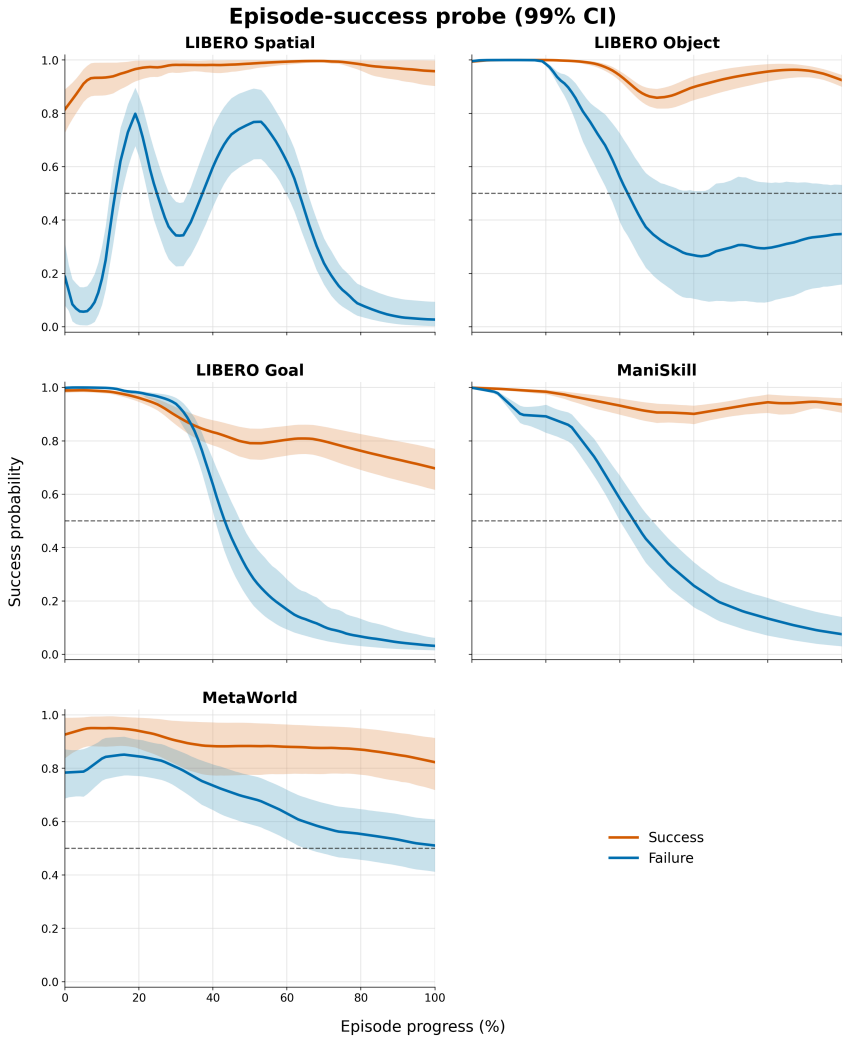


Figure 22: **Episode-level probe trajectories across different $\pi_{0.5}$ checkpoints.** Episode progress is normalized to 0–100% for each action horizon. Probe outputs from successful (orange) and failed (blue) episodes are interpolated over normalized progress, and we plot the mean prediction with 95% confidence intervals.

C.9 STEERING RESULTS

We examine whether steering along the shift directions can directly influence policy behavior. Steering the base policy along the learned shift directions produces substantial gains, supporting their causal relevance (Table 14). Here, steering is applied simultaneously to all sublayer–timestep pairs. Although the optimal coefficient α varies across benchmarks, performance generally improves as α approaches the optimum and declines once steering becomes too strong. These results show that shift vectors recover a substantial portion of the RL improvement when applied to the base policy.

Table 14: **Base-policy steering across different strengths in $\pi_{0.5}$.** Success rates (%) across steering coefficients α, where α scales the RL-induced shift vector before addition. Values in parentheses denote changes relative to the base policy. Best Recovery is the fraction of the RL improvement recovered by the best steering result.

Benchmark	Base	RL	Steering coefficient α				Best Recovery
			0.5	1.0	1.5	2.0	(%)
LIBERO Goal	82.8	93.6	**88.2** (+5.4)	84.4 (+1.6)	79.8 (−3.0)	73.2 (−9.6)	50.0
LIBERO Object	95.4	99.2	95.2 (−0.2)	97.6 (+2.2)	98.0 (+2.6)	**98.8** (+3.4)	89.5
LIBERO Spatial	85.0	96.6	86.4 (+1.4)	**92.0** (+7.0)	90.0 (+5.0)	90.2 (+5.2)	60.3
ManiSkill	42.2	89.1	48.4 (+6.2)	55.3 (+13.1)	57.5 (+15.3)	**63.8** (+21.6)	46.1
MetaWorld	42.8	69.2	56.4 (+13.6)	**59.4** (+16.6)	58.6 (+15.8)	57.2 (+14.4)	62.9