CONTRASTIVE CONCEPTOR ACTIVATION STEERING (COAST): UNLOCKING VISION-LANGUAGE-ACTION MODELS THROUGH HIDDEN STATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models leverage powerful perceptual priors from web-scale Vision-Language Model (VLM) pre-training, yet they remain surprisingly brittle in practice, frequently failing at simple robotic tasks. To mitigate this, we propose Contrastive Conceptor Activation Steering (COAST). COAST builds on the notion of a "conceptor", a linear operator that soft-projects data into the principal components of a target distribution. COAST uses conceptors to identify success-critical subspaces for a target robotic task from a few examples of success and failure rollouts. At inference time, it steers VLA latents into these identified success subspaces to improve task outcomes. Across three architecturally distinct neural policies (flow-matching VLA, autoregressive VLA, and Diffusion Policy), COAST improves absolute mean simulation and real-robot task success rate by over 20 and 40% respectively. The activation subspace geometry reveals that failure modes share substantial structure across tasks while success representations remain largely task-specific. When tasks share similar failure modes, this structure enables previously fitted conceptors to improve performance on new tasks without refitting. Ultimately, our results suggest that current VLAs retain substantial task-relevant knowledge in their latent representations, and that the action expert's decoding bottleneck could be mitigated by steering its residual stream toward task-relevant subspaces. COAST provides a lightweight, training-free path to unlocking these latent capabilities by steering the model towards its own "success" distributions. Project website: `coastvla34-pixel.github.io`

1 INTRODUCTION

Today's predominant paradigm for developing generalist robotic policies is to train Vision-Language-Action (VLA) models on multi-modal datasets at scale (Black et al., 2026; 2025; NVIDIA et al., 2025). VLAs have been shown to enable robots to interpret natural language commands in rich visual contexts and execute complex behaviors, exhibiting improved generalization (Kim et al., 2024; O'Neill et al., 2024). Yet, it is challenging to pre-train on the exhaustive distribution of all possible deployment conditions, and in practice these policies frequently fail to execute downstream tasks reliably (Liu et al., 2026; Kim et al., 2025). Approaches such as post-training and fine-tuning (Yuan et al., 2024; Wagenmaker et al., 2025) and contrastive representation regularization (Kim et al., 2025) have attempted to solve this problem. However, such remedies are costly and conceptually misaligned, as they attempt to relearn behaviors rather than trying to maximize the use of generalization ability that the model already learned from large-scale pre-training.

A natural point of comparison for VLAs is large language models (LLMs), where a growing body of work on *activation steering* has shown that lightweight, closed-form interventions on intermediate representations can recover correct behavior without retraining. Subramani et al. (2022) show that latent steering vectors can be extracted from frozen model weights. Turner et al. (2024) introduced activation addition using contrastive prompt pairs, Rimsky et al. (2024) generalized this to contrastive activation addition across positive and negative examples. These methods steer along a single direction, which limits their reach when the relevant signal is distributed across multiple dimensions. Zou et al. (2025) and Postmus & Abreu (2024) broadened these single-direction methods into a subspace-level framework.

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.

VLAs consist of a frozen vision-language backbone, which inherits rich perceptual and semantic representations from web-scale pretraining, and a separately trained action expert that decodes these representations into motor commands. We hypothesize that the action expert is the primary bottleneck. Its intermediate hidden states preserve much of the backbone's high-dimensional task understanding, but its final action outputs fail to fully exploit this information. If so, targeted interventions on the action expert's residual stream should be able to recover performance without modifying either the backbone or the expert's weights. Prior works have explored steering VLAs through their internal representations, but focus on the VLM backbone rather than the action expert. Häon et al. (2025) identify semantically meaningful neurons in the backbone's FFN layers through token projection, and Huben et al. (2024) and Khan et al. (2025) train sparse autoencoders on VLA hidden states to isolate useful features. However, these interventions steer individual backbone features and have not reported comprehensive improvements in overall task success rates. By contrast, COAST intervenes directly on the action expert's residual stream, where the decoding bottleneck resides.

Motivated by this, we introduce **COAST (Contrastive Conceptor Activation Steering)**, a plug-and-play framework that steers frozen VLA policies at inference time by applying a multiplicative gate to residual-stream activations. Conceptors were originally developed for recurrent networks (Jaeger, 2014) and recently applied to LLM steering (Postmus & Abreu, 2024; 2025; Triantafyllopoulos et al., 2026). We adapt this machinery to VLA action experts, using Boolean subspace algebra to construct contrastive conceptors from success and failure rollouts. The resulting operator isolates high-variance, outcome-critical directions while suppressing shared low-variance noise, and multiplicatively gates the residual stream during action generation. Unlike prior VLA activation steering methods that require learned auxiliary models, gradient computation, or per-feature interventions, COAST operates through closed-form operators with efficient parameter selections. We validate COAST across flow-matching VLAs, autoregressive VLAs, and Diffusion Policy, and find that it consistently and significantly outperforms existing steering approaches across all three architecture families.

Our main contributions are:

- **Lightweight Training-Free Policy Steering.** We apply conceptor-based subspace gating, previously used only in NLP, to VLA action experts for the first time. The resulting method, COAST, constructs a closed-form multiplicative gate from Boolean conceptor algebra over success and failure rollouts, improving mean simulation and real-robot task success rate by 20% and 40% respectively.
- **Mechanistic Insight.** Upon analysis to discover the sources of these large gains, we find that the eigenvalue structure of VLA action experts is well-suited for conceptor-based steering. Outcome-relevant representations occupy a low-rank but not rank-one subspace, explaining why conceptors outperform single-direction additive methods. We further show that COAST reshapes latent-space activations toward the success subspace.
- **Plug-And-Play Improvements.** We validate COAST across three architecturally distinct robot policy backbones (flow-matching VLAs, autoregressive VLAs, and Diffusion Policy), three simulation benchmarks of increasing complexity, cross-task conceptor transfer, and real-robot deployment, establishing that the method and the geometric structure it exploits are not specific to any single architecture, benchmark, task, or environment.

2 Related Work

Inference-time steering of robot policies. Recent work adapts frozen VLA policies at deployment through reward-guided steering (Liu et al., 2026), latent substitution with auxiliary world models (Das et al., 2025), or VLM-based planning and verification (Liu et al., 2026; Song et al., 2026; Shah et al., 2025; Wu et al., 2026). Other approaches instead modify the policy itself by contrastively updating representations (Kim et al., 2025), training explicit steering interfaces (Chen et al., 2026), or selectively fine-tuning model components (Mitra et al., 2025). Unlike these approaches, COAST does not require external foundation models, runtime search, or gradient-based optimization, operating instead via a closed-form linear intervention. We compare COAST against internal baselines, while still showing that it surpasses the evaluated LoRA-based SFT baseline in task performance (Sec. 4.2, App. B.1). Please refer to App. O for a more in-depth analysis.

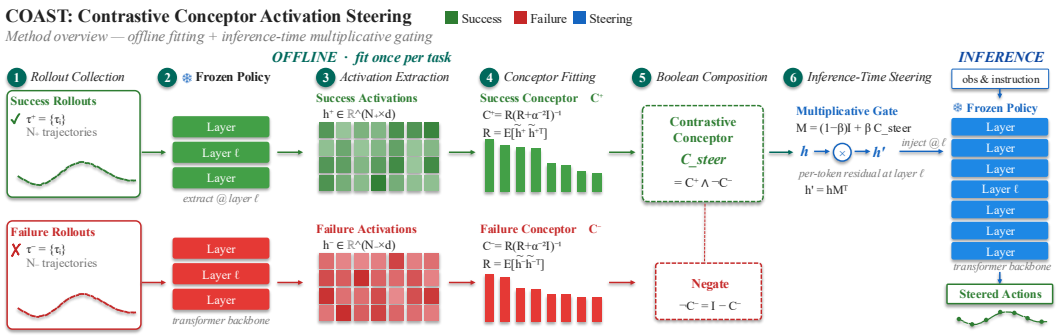


Figure 1: **Overview of COAST.** COAST steers a frozen robot policy at inference time by multiplicatively gating its residual stream with a contrastive conceptor fit from rollout activations. *Offline,* success and failure rollouts yield layer-ℓ activations, from which closed-form conceptors C^+ and C^- are fit and composed via Boolean subspace algebra into $C_{\text{steer}} = C^+ \wedge \neg C^-$. *Inference,* the residual h at layer ℓ is gated by $M = (1-\beta)I + \beta\, C_{\text{steer}}$ to produce $h' = hM^\top$, which flows through the rest of the frozen model. No gradient computation or weight modification is required.

Reward-based policy improvement. A separate line of work improves VLA and diffusion policies through reinforcement learning rather than activation steering. Wagenmaker et al. (2025) optimize a reward function in the latent noise space of a diffusion policy, while Pfrommer et al. (2025) train flow-matching policies via reward-weighted regression and group relative policy optimization. These methods require an explicit reward signal and gradient-based optimization of policy parameters or auxiliary modules. In contrast, COAST leaves the policy weights frozen and requires no reward beyond binary success/failure labels used to partition rollouts.

Conceptors. Conceptors are closed-form multiplicative gates on activation subspaces, introduced by Jaeger (2014) for recurrent networks. We introduce their technical details in Section 3.1. Postmus & Abreu (2024) applied them to LLM steering additively as steering vectors, and Postmus & Abreu (2025) extended the framework to optimal affine steering, showing that conceptor-based intervention outperforms additive methods and that Boolean composition of conceptors outperforms naive vector combination. In concurrent work, Triantafyllopoulos et al. (2026) apply contrastive conceptors to steer LLM semantic outputs, confirming the effectiveness of Boolean conceptor algebra for NLP tasks. We extend these insights to VLAs, applying contrastive subspace construction via Boolean negation and multiplicative residual-stream gating to VLA action experts. To our knowledge, this is the first use of conceptors for embodied policy steering.

3 Activation Steering with Contrastive Conceptor

COAST builds on existing conceptor algebra (Jaeger, 2014; Postmus & Abreu, 2024) and adapts it to VLA action experts. The key observation motivating this adaptation is that a VLA's hidden states during task execution contain both task-critical and task-irrelevant features. When comparing successes and failures, the irrelevant features remain largely unchanged and naturally exhibit low variance within collected samples, while task-relevant and outcome-critical states tend to manifest high variance. Conceptors are well-suited for this setting because they capture these highly variable and discriminative directions by suppressing shared low-variance noise, effectively steering the model toward the representations essential for task completion.

3.1 Background: Contrastive Conceptors For Identifying Task-Relevant State Space

Building Conceptors. A conceptor matrix C is a positive semi-definite linear operator that captures the principal directions and variances of a set of neural activations (Jaeger, 2014). Given an activation matrix $X \in \mathbb{R}^{N\times d}$ whose rows are individual hidden states, let $\tilde{X}$ denote the mean-centered matrix with rows $\tilde{x}_i = x_i - \bar{x}$, where $\bar{x} = \frac{1}{N}\sum_i x_i$. C minimizes a regularized reconstruction objective

$$\min_C \frac{1}{N}\|\tilde{X} - \tilde{X}C\|_F^2 \;+\; \alpha^{-2}\|C\|_F^2. \tag{1}$$

The first term ensures C preserves the variance structure of the activations, while the second, controlled by the aperture $\alpha > 0$, penalizes complexity. We follow the convention of Jaeger (2014), where larger α yields a less regularized conceptor that faithfully reconstructs the data, while smaller α favors a more compressed, generalized representation. The closed-form solution is $C = R(R + \alpha^{-2} I)^{-1}$, where $R = \tilde{X}^\top \tilde{X}/N$ is the covariance matrix. Crucially, this allows us to compute C directly via simple matrix multiplication and inversion, bypassing the need for gradient descent.

Conceptors as filters. Applying a conceptor C to a new sample x to produce $\tilde{x} = x^T C$ can be seen as "filtering" the sample. The eigenvalue structure of C makes this filtering behavior transparent. If R has eigenvalues $\{\lambda_i\}$, the corresponding eigenvalues of C are

$$\mu_i = \frac{\lambda_i}{\lambda_i + \alpha^{-2}}. \tag{2}$$

Directions of high variance ($\lambda_i \gg \alpha^{-2}$) receive $\mu_i \approx 1$ and pass through nearly unchanged, while low-variance directions ($\lambda_i \ll \alpha^{-2}$) are suppressed toward zero. The conceptor thus acts as a soft, variance-adaptive filter on the activation space. This way, a conceptor scales each direction differently based on the discovered subspace structure.

Contrastive Conceptor. A conceptor as described above captures the principal subspace of the activation distribution, but effective steering requires discriminating success from failure distributions. This can be captured by first computing separate conceptors C_{success} and C_{failure} for success and failure samples following Equation 1, and then appropriately *composing* them.

Conceptors compose conveniently under a soft Boolean algebra (Jaeger, 2014; Postmus & Abreu, 2024). In particular, NOT is defined as $\neg C = I - C$, and AND is defined as

$$A \wedge B \;=\; \left(A^{-1} + B^{-1} - I\right)^{-1}. \tag{3}$$

following the canonical conceptor algebra of Jaeger (2014) and Postmus & Abreu (2024). Intuitively, AND computes the soft intersection of two subspaces: directions that both operands retain receive high weight, while directions retained by only one are attenuated. Thus, a "contrastive conceptor" that separates success and failure states may be composed as:

$$C_{\text{steer}} = C_{\text{success}} \wedge \neg C_{\text{failure}}. \tag{4}$$

The AND-NOT operation retains directions present in the success subspace and absent from the failure subspace. Shared directions, which carry outcome-irrelevant computation, are suppressed. What remains is the thin residual subspace that differentiates the two outcomes.

3.2 Steering VLAs with Contrastive Conceptors

At inference time, we use the contrastive conceptor as a soft projection that reweights the action expert's activations without overwriting them, giving direct control over the model's latent state. This gently steers hidden states toward the success-associated subspace while suppressing directions linked to failure. Because the intervention acts only on a small, outcome-critical subspace, it is more efficient and less disruptive than manipulating the full representation.

Activation extraction. We focus on activations rather than other residual hidden states because they more cleanly capture the network's task-relevant dynamics as a structured, low-dimensional manifold (Jaeger, 2014). The VLA policies we evaluate share a common backbone-expert architecture in which a frozen vision-language model produces a conditioning representation and a trainable action expert generates actions through either flow-matching denoising (Black et al., 2025; NVIDIA et al., 2025) or autoregressive token prediction (Pertsch et al., 2025). Diffusion Policy (Chi et al., 2025) shares the flow-matching denoising architecture. We extract activations from the action expert's residual stream at a single layer ℓ, mean-pooling across the expert's suffix tokens (action tokens for $\pi_{0.5}$; state, future, and action tokens for GR00T N1.5) to obtain one vector $h \in \mathbb{R}^d$ per denoising or autoregressive step. We roll out N samples, where success and failure are determined either by simulation or by real world outcomes. Stacking N such vectors yields the matrix $X = [h_1; \ldots; h_N] \in \mathbb{R}^{N \times d}$ from which the conceptors in Section 3.1 are constructed. Details of the extraction procedure for each model family are in Appendix J.

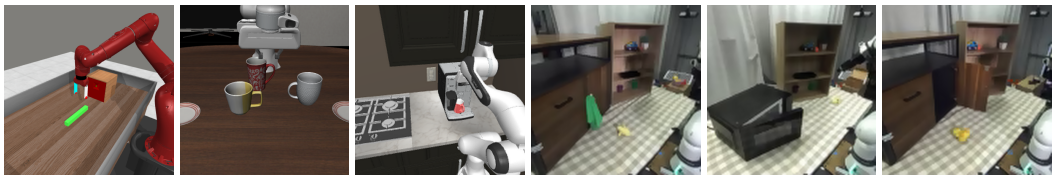


Figure 2: **Overview of Environment Setup.** We evaluate on three simulation benchmarks of increasing difficulty, MetaWorld ML45 (Yu et al., 2021), LIBERO-10 (Liu et al., 2023), and a select RoboCasa (Nasiriany et al., 2024) subset, plus three real-robot tasks on the DROID platform.

Multiplicative Gating. At inference time, we apply the contrastive conceptor as a multiplicative gate on the residual stream. The gating matrix is

$$M = (1 - \beta)\, I + \beta\, C_{\text{steer}}. \tag{5}$$

where $\beta \in [0, 1]$ controls the steering strength. Each residual activation h at layer ℓ is replaced by $h' = hM^\top$ before continuing through the remaining layers of the frozen model. The interpolation with the identity ensures that the gate acts as a soft projection. At $\beta = 0$ the forward pass is unmodified, at $\beta = 1$ the full conceptor is applied, and intermediate values blend the original and steered activations. The interpolation with the identity is important because even though C_{steer} is already a soft projection (its eigenvalues are fractional rather than binary), applying it at full strength can still be too aggressive. The β parameter provides an additional degree of control, allowing the gate to attenuate directions according to their discriminative value while preserving enough of the original activation to keep the policy's downstream computation stable.

Steering strategies. We evaluate three variants of conceptor steering. *Global* steering fits a single contrastive conceptor $C_{\text{steer}} = C_{\text{success}} \wedge \neg C_{\text{failure}}$ from activations pooled across all denoising steps and applies the same gate at every step. This is the simplest strategy and assumes the discriminative subspace is stable throughout the denoising process. *Per-step* steering instead fits a separate contrastive conceptor for each denoising step from activations collected at that step alone, allowing the gate to track shifts in the discriminative subspace as the action is progressively refined from noise to a clean action chunk. *Positive-only* steering sets $C_{\text{steer}} = C_{\text{success}}$, omitting the failure subtraction entirely. This variant requires only successful demonstrations and serves as a test of whether the contrastive operation is necessary or whether projecting toward the success subspace alone suffices.

We also introduce a highly efficient hyperparameter selection heuristic by identifying the optimal layer through conceptor quota and narrowing the aperture range via success-failure overlap. This recovers 93% of oracle performance while evaluating under 3-8% of the full configuration grid (App. I.2). Additionally, we test the latency of COAST and find it to be minimal and in-line with existing steering methods (App. P).

4 Experiments

In our experiments, we aim to evaluate the following questions. (1) **Performance.** Does COAST reliably improve task success across diverse VLA models and benchmarks? (2) **Mechanism & Interpretability.** How does conceptor steering geometrically reshape internal activations? (3) **Shared Failure Geometry.** Do tasks share failure-mode structure, and if so, can a conceptor fitted on one task improve a different task without refitting?

4.1 Experimental Setup

Environments. We evaluate across three simulation benchmarks and three real-world tasks chosen to span a range of task complexity, from short-horizon single-object manipulation to long-horizon multi-step reasoning in visually diverse scenes. Further setups can be found in App. K.

- **MetaWorld ML45** (Yu et al., 2021) comprises 45 tabletop robotic manipulation tasks spanning a broad spectrum of motor skills, each with randomized goal positions per episode.

- **LIBERO-10** (Liu et al., 2023) consists of 10 long-horizon manipulation tasks in tabletop kitchen scenes, each requiring multi-step object interactions such as picking, placing, and opening across semantically diverse objects including containers, utensils, and appliances.
- **RoboCasa** (Nasiriany et al., 2024) covers 7 atomic-seen tasks. Every episode re-samples the kitchen layout, object instances, placements, textures, and lighting, producing a combinatorially large state space that demands strong generalization across visually and physically diverse configurations unseen during training.
- **Real-world validation.** We use the DROID platform (Khazatsky et al., 2024), evaluated on three tasks: `Open Drawer`, `Close Microwave`, and `Put Duck in Cabinet` over 15 independent trials per task. Success is defined strictly by task completion: drawer fully extended, microwave door latched, or duck inside the cabinet. The full protocol and confidence intervals are in App. A.

Evaluated Policies. We evaluate several backbone policies across our environments:

$\pi_{0.5}$ (Black et al., 2025) (Metaworld, LIBERO, Robocasa), π_0-FAST (Pertsch et al., 2025) (Metaworld, LIBERO), GR00T N1.5 (NVIDIA et al., 2025) (Robocasa) and Diffusion Policy (Chi et al., 2025) (Robocasa). By showing that our method also works with DP, we demonstrate our method's scalability to the diffusion/flow-matching family of policies other than VLA models. All VLA models are also evaluated on real-world tasks. VLA models achieve very high success rates on MetaWorld and LIBERO when fully trained, so we use early fine-tuning checkpoints that produce the mix of successes and failures needed for contrastive fitting. For RoboCasa, the fully trained checkpoints for both GR00T N1.5 and Diffusion Policy exhibit sufficient failure rates, so no early stopping is required. We show that our method is robust across different checkpoints and parameters in App. C.3.

Baselines. We evaluate COAST against two activation steering baselines and one Supervised Fine-Tuning (SFT) baseline. We compare against Contrastive Activation Addition (CAA) (Rimsky et al., 2024) to represent single-direction additive steering, and Sparse Autoencoder (SAE) steering (Khan et al., 2025) to represent feature-based steering. Furthermore, we evaluate SFT with LoRA (Hu et al., 2022) on 30 on-policy rollouts per task, twice COAST's fitting budget, keeping only successful episodes. Implementation details for each baseline can be found in App. N.2, App. N.3, and App. N.1 respectively. Finally, aperture α and steering strength β are optimized on gathered rollouts (Sec. I.2).

Evaluation Protocol. For each task, we use a strict train/test split: 15 training rollouts to extract conceptors and select hyperparameters, and 30 held-out test rollouts with entirely unseen environment configurations for final evaluation (see App. K for details).

4.2 Does COAST reliably improve task success across diverse VLA models and benchmarks?

Across all model-benchmark pairs, COAST produces statistically significant improvements over the unsteered baseline, with contrastive strategies (global and per-step) consistently outperforming both additive (CAA) and feature-based (SAE) steering as well as supervised fine-tuning (Table 1). The gains are largest where the baseline is weakest and where success and failure subspaces overlap most heavily, consistent with the mechanistic analysis in Section 4.3. We summarize the key patterns per benchmark below and return to cross-benchmark observations at the end of this subsection.

MetaWorld ML45. Conceptor steering improves $\pi_{0.5}$ mean success rate from 0.57 to 0.82 under per-step contrastive steering ($+0.25$, $p < .001$), with individual tasks showing dramatic success rate recoveries: `pick-place-wall` from 0.20 to 1.00 and `stick-push` from 0.20 to 0.73 (Table 1; per-task detail in Table 26). π_0-FAST shows a consistent but smaller improvement, rising from 0.57 to 0.69 under global steering ($+0.13$, $p < .001$).

LIBERO-10. LIBERO produces the largest absolute gains of any benchmark. Per-step steering lifts $\pi_{0.5}$ from 0.43 to 0.80 ($+0.37$, $p < .001$), with the strongest recoveries on `LR5` ($0.07 \rightarrow 0.60$) and `KS3` ($0.53 \rightarrow 0.93$). π_0-FAST improves from 0.62 to 0.84 ($+0.23$, $p < .001$). Both CAA and

RoboCasa

Task	$\pi_{0.5}$							GR00T N1.5							Diffusion Policy					
	Baselines				*COAST*			*Baselines*				*COAST*			*Baselines*			*COAST*		
	Base	+SFT	+SAE	+CAA	+Glob.	+Per.	+Pos.	Base	+SFT	+SAE	+CAA	+Glob.	+Per.	+Pos.	Base	+SAE	+CAA	+Glob.	+Per.	+Pos.
Close Fridge	0.20‡	0.03	**0.47**	0.27	**0.47**	0.40	0.40	0.67	0.43	0.73	0.80	0.93	**1.00**	0.87	0.43	0.53	0.53	0.56	**0.60**	0.50
Coffee Mug	0.13	0.23	0.33	0.13	0.33	0.33	**0.53**	0.20‡	0.10	0.40	0.13	0.27	**0.33**	**0.33**	0.13	0.20	0.16	**0.23**	0.16	0.16
Open Drawer	0.53	0.47	0.60	0.53	**0.67**	0.53	**0.67**	0.53	0.57	0.67	0.73	**0.80**	0.53	0.67	0.10	0.20	0.23	**0.33**	0.30	0.23
Stand Mixer	0.67	0.30	0.53	0.53	0.73	**0.80**	0.53	0.60	0.77	0.73	0.80	0.80	0.80	**0.93**	0.63	**0.83**	0.63	**0.83**	**0.83**	**0.83**
PP Cabinet	0.60	0.50	0.67	0.67	**0.74**	**0.74**	0.67	0.73	0.63	0.73	0.53	**0.80**	**0.80**	0.73	0.30	0.53	0.40	0.46	**0.56**	0.46
PP Stove	0.33	0.43	0.53	0.53	0.53	**0.67**	0.60	0.73	0.60	**0.87**	**0.87**	**0.87**	0.75	**0.87**	0.10	0.10	0.10	**0.13**	**0.13**	0.10
Kettle	0.33‡	0.20	0.27	0.27	0.40	0.40	**0.53**	0.67	0.37	0.73	0.47	**0.80**	0.73	0.73	0.56	0.56	0.50	0.60	**0.66**	0.46
Mean	0.40	0.31	0.49	0.42	0.55	0.55	**0.56**	0.59	0.50	0.69	0.62	**0.75**	0.71	0.73	0.32	0.42	0.36	0.45	**0.46**	0.39
Δ	–	−0.09	+0.09	+0.02	+0.15	+0.15	+0.16	–	-0.09	+0.10	+0.03	+0.16	+0.12	+0.14	–	+0.10	+0.04	+0.13	+0.14	+0.07
p_t	–	.195	.172	.642	.002**	.009**	.044*	–	0.89	.006**	.690	.002**	.039*	.011*	–	.025*	.151	.004**	.006**	.123
p_z	–	.101	.211	.779	.014*	.014*	.009**	–	0.92	.113	.672	.006**	.046*	.014*	–	.134	.512	.058	.035*	.289

LIBERO-10

Task	$\pi_{0.5}$							π_0-FAST†						
	Baselines				*COAST*			*Baselines*				*COAST*		
	Base	+SFT	+SAE	+CAA	+Glob.	+Per.	+Pos.	Base	+SFT	+SAE	+CAA	+Glob.	+Per.	+Pos.
Stove+Moka	0.53	0.63	0.67	0.67	**0.93**	0.87	0.80	0.80	0.73	0.80	0.73	**1.00**	0.87	0.80
Bowl+Drawer	0.40	0.20	0.80	0.47	0.73	0.80	**0.87**	0.67	0.60	0.87	0.73	**0.93**	**0.93**	0.80
Mug+Micro	0.13‡	0.10	0.20	0.13	0.40	**0.60**	0.33	0.47	0.47	0.67	**0.80**	0.73	**0.80**	0.73
Two Mokas	0.20‡	0.07	0.13	0.07	0.47	**0.53**	0.13	0.33	0.10	0.33	0.27	**0.53**	**0.53**	0.40
Soup+Cheese	0.80	0.43	0.73	0.80	0.93	**1.00**	0.93	0.67	0.73	0.80	0.80	0.87	0.87	**1.00**
Soup+Tomato	0.40	0.53	**0.80**	0.53	**0.80**	**0.80**	0.60	0.67	0.77	**0.93**	0.80	0.87	**0.93**	0.73
Cheese+Butter	0.60	0.67	0.80	0.60	**0.93**	**0.93**	0.87	**1.00**	0.90	–	–	–	–	**1.00**
Mugs+Plates	0.07‡	0.13	0.33	0.07	**0.60**	**0.60**	0.20	0.60	0.73	**0.87**	0.80	**0.87**	**0.87**	0.80
Mug+Choc	0.53	0.57	0.73	0.60	**0.87**	**0.87**	0.67	0.73	0.77	0.87	0.67	**0.93**	**0.93**	0.87
Book+Caddy	0.67	0.70	0.87	0.80	0.93	**1.00**	0.93	0.60	0.43	0.73	0.80	**0.87**	**0.87**	0.73
Mean	0.43	0.40	0.61	0.47	0.76	**0.80**	0.63	0.65	0.62	0.76	0.71	**0.84**	**0.84**	0.79
Δ	–	−0.03	+0.17	+0.04	+0.33	+0.37	+0.20	–	−0.03	+0.15	+0.10	+0.23	+0.23	+0.13
p_t	–	.562	.009**	.157	< .001***	< .001***	.001**	–	.425	.002**	.074	< .001***	< .001***	.004**
p_z	–	.118	.003**	.487	< .001***	< .001***	< .001***	–	0.672	.009**	.094	< .001***	< .001***	.009**

MetaWorld ML45

Task	$\pi_{0.5}$							π_0-FAST						
	Baselines				*COAST*			*Baselines*				*COAST*		
	Base	+SFT	+SAE	+CAA	+Glob.	+Per.	+Pos.	Base	+SFT	+SAE	+CAA	+Glob.	+Per.	+Pos.
coffee-push	0.80	0.63	**1.00**	0.87	**1.00**	**1.00**	**1.00**	0.88	0.87	0.88	**1.00**	0.94	0.94	0.94
push	0.93	0.80	0.93	0.93	**1.00**	**1.00**	0.93	0.75	0.83	0.88	0.75	**0.94**	0.88	0.75
pick-place	0.87	0.87	0.80	0.87	**1.00**	**1.00**	0.80	0.69	0.67	0.62	0.75	**0.88**	**0.88**	0.81
plate-slide-back	0.60	0.37	0.53	0.53	**0.93**	**0.93**	0.73	0.88	0.93	0.56	0.75	0.88	0.88	**0.94**
faucet-close	0.80	0.67	**1.00**	0.93	**1.00**	0.93	0.93	0.44	0.80	**1.00**	0.50	0.69	0.69	0.81
pick-place-wall	0.20	0.23	0.33	0.40	0.87	**1.00**	0.47	0.69	0.50	**0.88**	0.69	0.81	0.75	0.75
reach	0.93	0.73	0.93	**1.00**	0.93	0.93	**1.00**	**0.81**	0.77	0.69	**0.81**	**0.81**	**0.81**	**0.81**
coffee-pull	0.93	0.67	**1.00**	**1.00**	0.93	0.93	**1.00**	0.62	0.60	0.62	0.50	**0.69**	0.62	**0.69**
disassemble	0.60	0.10	0.80	0.73	**0.93**	**0.93**	0.87	0.56	0.70	**0.81**	0.56	0.69	0.56	0.75
stick-push	0.20	0.20	0.40	0.60	**0.80**	0.73	0.40	0.81	0.90	**1.00**	0.81	0.88	0.88	0.88
Mean	0.69	0.53	0.77	0.79	**0.94**	**0.94**	0.81	0.71	0.76	0.79	0.71	**0.82**	0.79	0.81
Δ	–	−0.16	+0.08	+0.10	+0.25	+0.25	+0.13	–	+0.05	+0.08	+0.00	+0.11	+0.08	+0.10
p_t	–	.011*	.064	.039*	.008**	.012*	.006**	–	.358	.315	.968	.003**	.023*	.018*
p_z	–	< .001***	.029*	.004**	< .001***	< .001***	< .001***	–	0.0118	.092	.300	< .001***	< .001***	< .001***

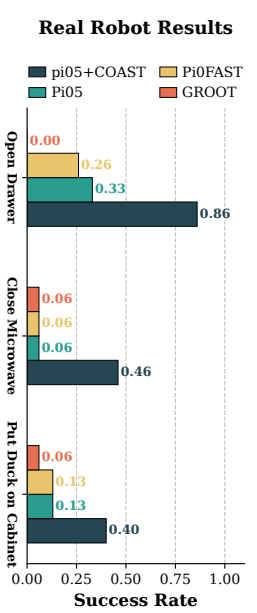


Table 1: **Contrastive conceptors deliver the largest and most consistent gains across every model–benchmark pair.** COAST strategies (+Glob., +Per., +Pos.; Sec. 3.2) versus +CAA, +SAE, and +SFT, on 30 held-out test rollouts per task and condition. Δ: mean gain over the unsteered baseline. p_t (primary): paired t-test across tasks; p_z: pooled two-proportion z-test; $^*p < 0.05$, $^{**}p < 0.01$, $^{***}p < 0.001$. **Bold**: best; underline: second best. †Zero failures under π_0-FAST, so no contrastive conceptor; column means over 9 tasks. ‡At most three successful fitting episodes. **(Right)** Real-robot success rates over 15 trials per task and condition. Task names, baseline settings, corrected p-values, and confidence intervals: App. H.

SFT falter here, showing only minimal gains ($+0.04$ and -0.03 for $\pi_{0.5}$, respectively); We attribute this to long-horizon dynamics, where small early deviations compound across steps and explicitly suppressing failure modes prevents this accumulation.

RoboCasa. RoboCasa requires generalization across randomized placements, textures, and cluttered scenes—the hardest setting. Steering nonetheless yields significant gains across models. $\pi_{0.5}$ reaches 0.56 ($+0.17$, $p_z = .009$) and GR00T N1.5 achieves 0.75 ($+0.16$, $p_z = .006$), showing sharp recoveries on `Stand Mixer`. While Diffusion Policy also improves (0.32→0.46, $+0.14$), its absolute gains are slightly lower than those of the VLA models; this suggests that while steerable success geometry exists in continuous diffusion representations, the inherited caches from VLM-backbones of VLAs might offer more robust decodability for activation steering. That all strategies remain significant confirms this geometry persists even under high visual variability.

Real World. To evaluate whether our method transfers beyond simulation, we test on three real-robot manipulation tasks: `Open Drawer`, `Close Microwave`, and `Put Duck in Cabinet`. We highlight that these tasks each have very different dynamics, and also were done on $\pi_{0.5}$-DROID policy that was not fine-tuned on any demos. Across all three tasks, $\pi_{0.5}$+COAST achieves the highest success rate with a 40% average improvement, with the largest gain on the challenging `Open Drawer` task. We also note that with just a single successful rollout, we were able

to increase the success rate to 46% on the `Close Microwave` task. The full protocol, per-task hyperparameters, and Wilson 95% confidence intervals are reported in App. A; with 15 trials per condition these intervals are wide. Visual examples of rollouts are in Fig. 7.

Comparison against baselines. COAST outperforms CAA because CAA applies a strictly additive shift along a single direction, while conceptors act as subspace-aware filters that selectively scale task-relevant directions and suppress irrelevant ones. COAST also outperforms SAE steering: SAE methods select features in a sparse basis but collapse them back into a single additive direction, inheriting CAA's lack of subspace-aware scaling and adding a reconstruction bottleneck on top. SFT underperforms the base policy on five environments because fine-tuning on a strictly limited budget of 30 rollouts per task often induces overfitting, degrading the model's pre-trained robustness; scaling SFT to 200 trajectories and 1,000 steps does not close the gap (App. B.1). Under a matched train/test tuning protocol with wider grids, COAST's margin over CAA and SAE grows (App. B.2).

Cross-benchmark patterns. Three patterns hold across all benchmarks and architectures. First, contrastive steering (global or per-step) consistently outperforms positive-only steering, confirming that the failure-subtraction operation contributes meaningfully beyond simply projecting toward success. Second, per-step steering matches or exceeds global steering on long-horizon tasks (LIBERO, RoboCasa) but offers little additional benefit on short-horizon tasks (MetaWorld), suggesting that the discriminative subspace shifts during denoising primarily in settings where multi-step reasoning is required. Third, gains scale inversely with baseline success rate. Tasks where the unsteered policy struggles most show the largest absolute improvements, while near-ceiling tasks show more moderate improvements.

4.3 How does conceptor steering geometrically reshape internal activations?

The gains reported in Section 4.2 raise a natural question. What property of the residual stream lets a single closed-form gate produce double-digit success-rate improvements? We identify three geometric facts about the action expert's hidden states (Fig. 3 and Fig. 4) to shed light on the effect.

Outcome-relevant computation is low-rank, but not rank-one. Although the residual stream of the $\pi_{0.5}$ model operates in a 1024-dimensional space, the eigenvalue spectra of both the success ($\mathbf{C}^+$) and failure ($\mathbf{C}^-$) conceptors decay sharply across all steered MetaWorld ML45 tasks (Figure 3A). By applying the Boolean contrastive operation ($\mathbf{C}^+ \wedge \neg\mathbf{C}^-$), the conceptor cancels out dimensions shared by both outcomes, reducing the effective steering subspace to roughly one percent of the total hidden dimension. This low-rank structure enables a closed-form operator to isolate success-relevant directions without disrupting the policy's overall computation. However, this discriminative subspace is strictly multi-dimensional, rather than rank-one. Additive steering along a single mean-difference vector recovers only partial gains (illustrated by the CAA baseline in Table 1), confirming that rank-one methods cannot fully capture this compact, multi-dimensional outcome geometry.

Subspace overlap predicts steering efficacy. The magnitude of steering gain depends heavily on how much the success and failure subspaces overlap. Plotting per-task improvement (ΔSR) against the success-failure overlap $\mathrm{sim}(\mathbf{C}^+, \mathbf{C}^-) = \mathrm{tr}(\mathbf{C}^+\mathbf{C}^-)/\sqrt{\mathrm{tr}((\mathbf{C}^+)^2)\,\mathrm{tr}((\mathbf{C}^-)^2)}$ yields a strong positive correlation ($\rho = 0.59$, $p = 0.002$; Figure 3B). When success and failure subspaces are nearly disjoint, the policy has already separated the two outcomes geometrically and there is little contrastive signal to amplify. When they overlap heavily, the thin residual directions distinguishing them are exactly what the Boolean AND-NOT operation isolates. Because overlap requires only a single matrix product and no rollouts, it also serves as a cheap diagnostic for predicting where steering will be most effective (App. I.2).

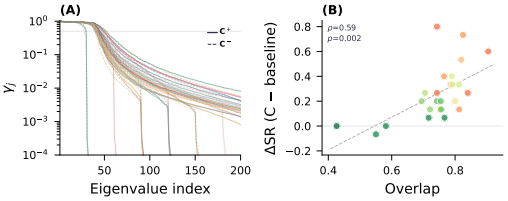


Figure 3: **The contrastive subspace is low-rank and predictive. (A)** Eigenvalue spectra of success ($\mathbf{C}^+$, solid) and failure ($\mathbf{C}^-$, dashed) conceptors on MetaWorld ML45 decay rapidly. **(B)** Per-task steering gain ΔSR against overlap $\mathrm{sim}(\mathbf{C}^+, \mathbf{C}^-)$. Spearman $\rho = 0.59$, $p = 0.002$.

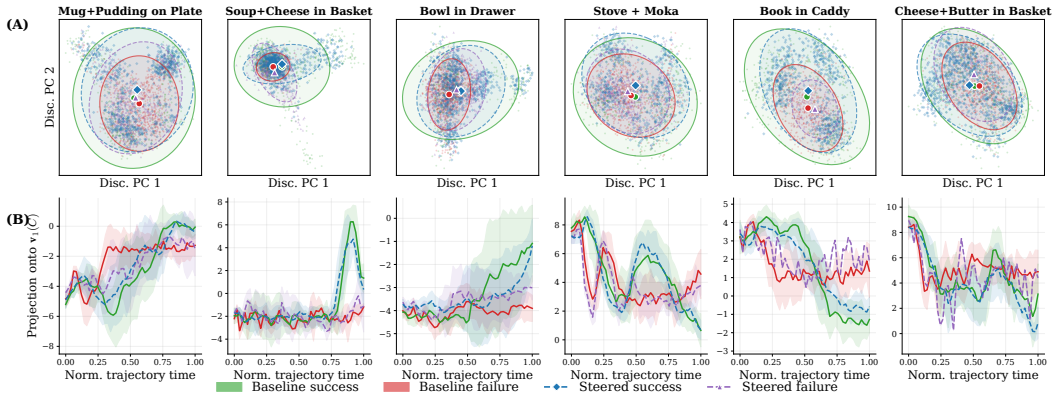


Figure 4: **Conceptor steering pulls activations toward the success region.** **(A)** Per-step activations projected onto the top two eigenvectors of $\mathbf{C}_{\text{steer}}$, the subspace maximally separating baseline success from failure. Solid ellipses: baseline 2σ regions; dashed: steered; large markers: centroids. **(B)** Projection onto $\mathbf{v}_1(\mathbf{C}_{\text{steer}})$ over normalized trajectory time (means $\pm 1\sigma$).

(a) $\pi_{0.5}$ LIBERO				(b) $\pi_{0.5}$ RoboCasa			
Target	**Self**	**Top-1**	**Top-2**	**Target**	**Self**	**Top-1**	**Top-2**
Stove+Moka	+0.40	**Two Mokas (PS) +0.60**	Soup+Tomato (G) +0.33	CloseFridge	+0.27	PP_Stove (PS) +0.07	OpenDrawer (PS) −0.20
Bowl+Drawer	+0.40	Soup+Tomato (PS) +0.20	Mug+Pudding (PS) +0.14	CoffeeSetup	+0.20	Kettle (G) +0.00	OpenDrawer (PS) −0.20
Mug+Micro	+0.47	Two Mokas (PS) +0.20	Mug+Pudding (G) −0.13	OpenDrawer	+0.14	**CoffeeSetup (PS) +0.60**	CoffeeSetup (PS) +0.47
Two Mokas	+0.40	Mugs+Plates (PS) +0.40	Mug+Pudding (PS) −0.06	StandMixer	+0.13	**CoffeeSetup (G) +0.54**	**CoffeeSetup (G) +0.53**
Soup+Cheese	+0.20	**Two Mokas (G) +0.67**	Soup+Tomato (G) +0.47	PP_Cabinet	+0.13	**CoffeeSetup (PS) +0.54**	CoffeeSetup (G) +0.47
Soup+Tomato	+0.40	Bowl+Drawer (PS) +0.27	Bowl+Drawer (PS) +0.27	PP_Stove	+0.34	CoffeeSetup (G) +0.40	CoffeeSetup (PS) +0.34
Cheese+Butter	+0.33	**Mug+Micro (G) +0.80**	**Soup+Tomato (PS) +0.60**	Kettle	+0.07	CoffeeSetup (PS) +0.14	CloseFridge (PS) +0.07
Mugs+Plates	**+0.60**	**Mug+Micro (G) +0.60**	Soup+Tomato (PS) +0.40				
Mug+Pudding	+0.34	**Mug+Micro (G) +0.67**	Soup+Tomato (PS) +0.40				
Book+Caddy	+0.33	**Soup+Tomato (PS) +0.53**	Soup+Tomato (G) +0.47				

Table 2: Transfer experiment across benchmarks. For each target task, we report the best self-steered result (Self) and the top-2 transfer sources. Each transfer cell shows *source (strategy) SR delta*, indicating the change in success rate compared to the target task's unsteered baseline. G = global strategy, PS = per-step strategy. Values with SR delta $\geq +0.50$ are shown in **bold**. LIBERO task names follow Table 1.

Steering reshapes the activation geometry toward success. Figure 4 visualizes the intervention's geometric effect. Panel A projects per-step activations onto the top two eigenvectors of each task's $\mathbf{C}_{\text{steer}}$, the directions along which success and failure are maximally separated. Green and red points denote individual steered-success and steered-failure activations respectively. The key observation is not simply that steered activations fall within the baseline-success ellipse, but that their *centroids shift* toward the baseline-success centroid and away from the baseline-failure centroid. In Panel B, this shift is visible over trajectory time. Steered successes consistently track the baseline-success mean throughout the rollout, while steered failures either sustain a partial shift or revert to the baseline-failure trajectory entirely. The fact that the physical outcome (success or failure) closely tracks which distribution the activations align with confirms that these geometric regions are not merely descriptive but linked to task execution.

4.4 Do tasks share failure geometry, and does it enable cross-task steering?

The self-steering results above prompt a natural follow-up question about the structure of the representations COAST exploits. Examining the activation geometry across tasks reveals an asymmetry. Success activations cluster tightly and uniformly across tasks, while failure activations spread into distinct, task-specific regions (Figure 5, left). In other words, many tasks share overlapping failure modes even though their success representations remain largely disjoint.

This asymmetry has a practical consequence. Because the contrastive conceptor from Section 3 acts subtractively along failure directions, a conceptor fitted on one source task should also help a different target task whenever their failure subspaces overlap. We test this by fitting conceptors on a source task and applying them to a different target within the same benchmark (Table 2). Transferred conceptors retain most of the gain of the self-fit configuration, and in several cases a transferred conceptor matches or exceeds it.

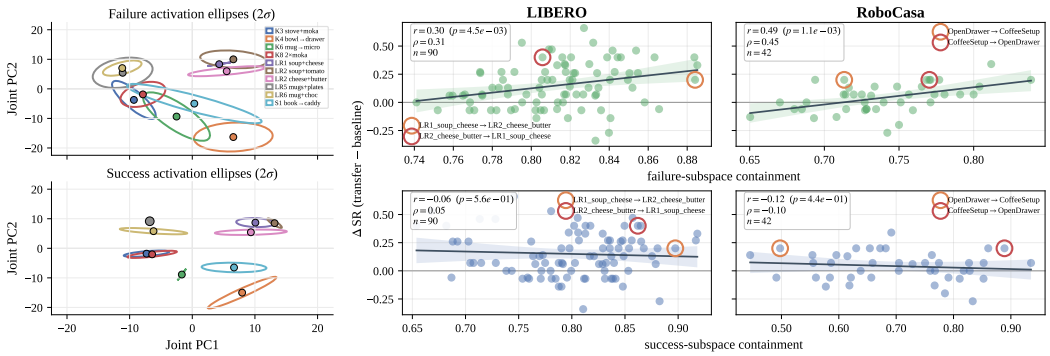


Figure 5: **Tasks share failure geometry but not success geometry, enabling cross-task steering.** *Left:* Joint PCA of layer-11 activations on $\pi_{0.5}$ LIBERO, 2σ ellipses per task. Failure activations (top) spread into task-specific regions whose overlap varies; success activations (bottom) cluster uniformly across tasks. *Right:* Each dot is one source→target pair on LIBERO or RoboCasa. Failure-subspace containment $\mathrm{tr}(C^f_{\mathrm{src}} C^f_{\mathrm{tgt}})/\mathrm{tr}((C^f_{\mathrm{src}})^2)$ correlates with transfer success-rate gain (top; r=0.30 and 0.49, p<0.005); success containment does not (bottom; $|r|$<0.13, p>0.4).

We quantify this effect by measuring failure-subspace containment $\mathrm{tr}(C^f_{\mathrm{src}} C^f_{\mathrm{tgt}})/\mathrm{tr}((C^f_{\mathrm{src}})^2)$ between source and target conceptors at L=11. Across off-diagonal source/target pairs, containment correlates with the transfer success-rate gain over baseline (Pearson r=0.30, p=4.5e−3 on LIBERO; r=0.49, p=1.1e−3 on RoboCasa; Figure 5, top right). The analogous success-subspace containment shows no such relationship (both $|r|$<0.13, p>0.4; bottom right), ruling out the simpler explanation that tasks just share geometry in general. This confirms that shared failure structure, not shared success structure, is the driver of cross-task steering gains. App. D reports transfer with no source selection at all (a uniform average of all other tasks' conceptors) and states what data an oracle-free selection rule requires.

5 Limitations and Conclusion

Limitations. Contrastive conceptors require both success and failure rollouts, which means uniformly successful tasks might fall back to positive-only steering, which is safe but could be less effective than its contrastive counterpart (App. F.6). Furthermore, tasks with nearly disjoint success and failure subspaces show smaller gains, as the overlap-efficacy relationship predicts. Lastly, whether the geometric hyperparameter selection generalizes beyond the architectures tested here remains open.

Conclusion. Frozen action experts in VLA models already encode the information needed to distinguish successful from failed behavior, and COAST surfaces it with a simple closed-form operator, improving mean simulation and real-robot success rates by over 20% and 40% respectively. The geometric structure that enables steering, low-rank subspaces with contrastively separable success and failure directions, persists across flow-matching, autoregressive, and Diffusion Policy architectures, and shared failure geometry lets a conceptor fitted on one task improve related tasks without refitting. VLA execution failures can thus be mitigated without retraining when the policy's internal representations already encode what success requires.

Broader impact. COAST lowers the cost of adapting frozen robot policies to new tasks, replacing fine-tuning with a small number of rollouts and a closed-form computation. This accessibility is a double-edged sword. Because steering acts on internal activations rather than outputs, a steered policy is difficult to distinguish from an unmodified one based on action traces alone, and the same operator could be constructed to steer toward failure-aligned behavior. Two properties bound the intervention: the steering matrix $M = (1-\beta)I + \beta\, C_{\mathrm{steer}}$ has eigenvalues in $[0, 1]$ and spectral norm at most one, so it can only attenuate directions of the residual stream and never amplifies activations, and the intervention is fully specified by a stored matrix that can be logged. We recommend that deployments log applied conceptors as part of the policy specification and pair activation steering with output-level monitoring and standard physical safeguards.

AI USE STATEMENT

In this work, we used generative AI tools for editing the manuscript text (grammar, spelling, and word choice). We have not used generative AI tools for research ideation, literature search, experimental design, code generation, data analysis, or the generation of results, figures, or tables, and the remaining tasks with required disclosure are not applicable to this work. We have reviewed all AI-assisted edits for accuracy and faithfulness to the underlying results. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In Joseph Lim, Shuran Song, and Hae-Won Park (eds.), *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 27–30 Sep 2025. URL https://proceedings.mlr.press/v305/black25a.html.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2026. 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, 2026. URL https://arxiv.org/abs/2603.05487.

William Chen, Jagdeep Singh Bhatia, Catherine Glossop, Nikhil Mathihalli, Ria Doshi, Andy Tang, Danny Driess, Karl Pertsch, and Sergey Levine. Steerable vision-language-action policies for embodied reasoning and hierarchical control, 2026. URL https://arxiv.org/abs/2602.13193.

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.

Satyajeet Das, Darren Chiu, Zhehui Huang, Lars Lindemann, and Gaurav S. Sukhatme. Latent activation editing: Inference-time refinement of learned policies for safer multirobot navigation, 2025. URL https://arxiv.org/abs/2509.20623.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders, 2024. URL https://arxiv.org/abs/2406.04093.

Bear Häon, Kaylene Stocking, Ian Chuang, and Claire Tomlin. Mechanistic interpretability for steering vision-language-action models, 2025. URL https://arxiv.org/abs/2509.00328.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

Robert Huben, Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=F76bwRSLeK.

Herbert Jaeger. Controlling recurrent neural networks by conceptors, 2014. URL https://arxiv.org/abs/1403.3369.

Momin Ahmad Khan, Novak Boskov, Fatima M. Anwar, and Manzoor A. Khan. Controlling vision–language–action policies through sparse latent directions. In *Mechanistic Interpretability Workshop at NeurIPS 2025*, 2025. URL https://openreview.net/forum?id=wtf3ww1EOL.

Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama, Lawrence Yunliang Chen, Kirsty Ellis, Peter David Fagan, Joey Hejna, Masha Itkina, Marion Lepert, Yecheng Jason Ma, Patrick Tree Miller, Jimmy Wu, Suneel Belkhale, Shivin Dass, Huy Ha, Arhan Jain, Abraham Lee, Youngwoon Lee, Marius Memmel, Sungjae Park, Ilija Radosavovic, Kaiyuan Wang, Albert Zhan, Kevin Black, Cheng Chi, Kyle Beltran Hatch, Shan Lin, Jingpei Lu, Jean Mercat, Abdul Rehman, Pannag R Sanketi, Archit Sharma, Cody Simpson, Quan Vuong, Homer Rich Walke, Blake Wulfe, Ted Xiao, Jonathan Heewon Yang, Arefeh Yavary, Tony Z. Zhao, Christopher Agia, Rohan Baijal, Mateo Guaman Castro, Daphne Chen, Qiuyu Chen, Trinity Chung, Jaimyn Drake, Ethan Paul Foster, Jensen Gao, David Antonio Herrera, Minho Heo, Kyle Hsu, Jiaheng Hu, Donovon Jackson, Charlotte Le, Yunshuang Li, Xinyu Lin, Zehan Ma, Abhiram Maddukuri, Suvir Mirchandani, Daniel Morton, Tony Khuong Nguyen, Abigail O'Neill, Rosario Scalise, Derick Seale, Victor Son, Stephen Tian, Emi Tran, Andrew E. Wang, Yilin Wu, Annie Xie, Jingyun Yang, Patrick Yin, Yunchu Zhang, Osbert Bastani, Glen Berseth, Jeannette Bohg, Ken Goldberg, Abhinav Gupta, Abhishek Gupta, Dinesh Jayaraman, Joseph J Lim, Jitendra Malik, Roberto Martín-Martín, Subramanian Ramamoorthy, Dorsa Sadigh, Shuran Song, Jiajun Wu, Michael C. Yip, Yuke Zhu, Thomas Kollar, Sergey Levine, and Chelsea Finn. DROID: A large-scale in-the-wild robot manipulation dataset. In *RSS 2024 Workshop: Data Generation for Robotics*, 2024. URL https://openreview.net/forum?id=Ml2pTYLNLi.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model, 2024. URL https://arxiv.org/abs/2406.09246.

Taeyoung Kim, Jimin Lee, Myungkyu Koo, Dongyoung Kim, Kyungmin Lee, Changyeon Kim, Younggyo Seo, and Jinwoo Shin. Contrastive representation regularization for vision-language-action models, 2025. URL https://arxiv.org/abs/2510.01711.

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

Shuo Liu, Ishneet Sukhvinder Singh, Yiqing Xu, Jiafei Duan, and Ranjay Krishna. Vls: Steering pretrained robot policies via vision-language models, 2026. URL https://arxiv.org/abs/2602.03973.

Chancharik Mitra, Yusen Luo, Raj Saravanan, Dantong Niu, Anirudh Pai, Jesse Thomason, Trevor Darrell, Abrar Anwar, Deva Ramanan, and Roei Herzig. Mechanistic finetuning of vision-language-action models via few-shot demonstrations, 2025. URL https://arxiv.org/abs/2511.22697.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots, 2024. URL https://arxiv.org/abs/2406.02523.

NVIDIA, :, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng,

and Yuke Zhu. Gr00t n1: An open foundation model for generalist humanoid robots, 2025. URL https://arxiv.org/abs/2503.14734.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, and et al Gupta. Open x-embodiment: Robotic learning datasets and rt-x models : Open x-embodiment collaboration0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903, 2024. doi: 10.1109/ICRA57147.2024.10611477.

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, 2025. URL https://arxiv.org/abs/2501.09747.

Samuel Pfrommer, Yixiao Huang, and Somayeh Sojoudi. Reinforcement learning for flow-matching policies. *arXiv preprint arXiv:2507.15073*, 2025. URL https://arxiv.org/abs/2507.15073.

Joris Postmus and Steven Abreu. Steering large language models using conceptors: Improving addition-based activation engineering. In *MINT: Foundation Model Interventions*, 2024. URL https://openreview.net/forum?id=gyAnAq16HC.

Joris Postmus and Steven Abreu. Steering large language models using conceptors: Improving addition-based activation engineering, 2025. URL https://arxiv.org/abs/2410.16314.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering llama 2 via contrastive activation addition. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15504–15522, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.828. URL https://aclanthology.org/2024.acl-long.828/.

Ameesh Shah, William Chen, Adwait Godbole, Federico Mora, Sanjit A. Seshia, and Sergey Levine. Learning affordances at inference-time for vision-language-action models, 2025. URL https://arxiv.org/abs/2510.19752.

Yunzhou Song, Long Le, Yong-Hyun Park, Jie Wang, Junyao Shi, Lingjie Liu, Jiatao Gu, Eric Eaton, Dinesh Jayaraman, and Kostas Daniilidis. Omniguide: Universal guidance fields for enhancing generalist robot policies. *arXiv preprint arXiv:2603.10052*, 2026.

Nishant Subramani, Nivedita Suresh, and Matthew Peters. Extracting latent steering vectors from pretrained language models. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 566–581, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.48. URL https://aclanthology.org/2022.findings-acl.48/.

Ilias Triantafyllopoulos, Young-Min Cho, Ren Tao, Miranda Muqing Miao, Sunny Rai, Lyle Ungar, Sharath Chandra Guntuku, Neville Ryant, and João Sedoc. Conceptors for semantic steering, 2026. URL https://arxiv.org/abs/2605.04980.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering, 2024. URL https://arxiv.org/abs/2308.10248.

Andrew Wagenmaker, Mitsuhiko Nakamoto, Yunchu Zhang, Seohong Park, Waleed Yagoub, Anusha Nagabandi, Abhishek Gupta, and Sergey Levine. Steering your diffusion policy with latent space reinforcement learning, 2025. URL https://arxiv.org/abs/2506.15799.

Yilin Wu, Anqi Li, Tucker Hermans, Fabio Ramos, Andrea Bajcsy, and Claudia Pérez-D'Arpino. Do what you say: Steering vision-language-action models via runtime reasoning-action alignment verification, 2026. URL https://arxiv.org/abs/2510.16281.

Siyuan Yang, Yang Zhang, Haoran He, Ling Pan, Xiu Li, Chenjia Bai, and Xuelong Li. Steering vision-language-action models as anti-exploration: A test-time scaling approach. *arXiv preprint arXiv:2512.02834*, 2025.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021. URL https://arxiv.org/abs/1910.10897.

Xiu Yuan, Tongzhou Mu, Stone Tao, Yunhao Fang, Mengke Zhang, and Hao Su. Policy decorator: Model-agnostic online refinement for large policy model, 2024. URL https://arxiv.org/abs/2412.13630.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, Alexander Pan, Xuwang Yin, Mantas Mazeika, Ann-Kathrin Dombrowski, Shashwat Goel, Nathaniel Li, Michael J. Byun, Zifan Wang, Alex Mallen, Steven Basart, Sanmi Koyejo, Dawn Song, Matt Fredrikson, J. Zico Kolter, and Dan Hendrycks. Representation engineering: A top-down approach to ai transparency, 2025. URL https://arxiv.org/abs/2310.01405.

This appendix is organized so that the material most relevant to assessing the main claims comes first. App. A reports the complete real-robot protocol, per-task calibration data, selected hyperparameters, and Wilson 95% confidence intervals. App. B strengthens the fine-tuning baseline (200 trajectories, up to 1,000 steps) and re-runs the CAA and SAE baselines under the same train/test protocol as COAST. App. C tests whether COAST only helps undertrained policies, using a headroom analysis, the fully trained checkpoints, and a Diffusion Policy epoch sweep. App. D gives an oracle-free rule for cross-task transfer and an average-source transfer experiment. App. E reports the primary significance test and multiple-comparison corrections. App. F quantifies the calibration cost of COAST, the relaxed data regimes (positive-only, failure-only, single-success, transfer), the eligibility criteria for contrastive fitting, and the composition of the fitting data. The remaining appendices contain operator ablations (App. G), per-task results (App. H), hyperparameter sweeps and selection (App. I), and reproducibility details: activation extraction (App. J), rollout collection (App. K), conceptor computation (App. L), checkpoints and training (App. M), baseline implementations (App. N), an extended comparison of adaptation paradigms (App. O), and inference overhead (App. P).

A Real-World Evaluation: Protocol, Calibration Data, and Confidence Intervals

We used the DROID robot setup (Khazatsky et al., 2024), which consists of a 7 DoF Franka Emika Panda Robot Arm, a Robotiq 2F-85 parallel-jaw gripper, a wrist-mounted ZED Mini RGB-D camera and two side-mounted ZED 2 stereo cameras. The DROID set-up enables the usage of the generalist VLA policies, specifically the $\pi_{0.5}$-DROID checkpoint (Black et al., 2025). Figure 6 shows the hardware configuration, and Table 3 summarizes the full protocol. The policy is the released $\pi_{0.5}$-DROID checkpoint and is not fine-tuned on any demonstrations of the evaluated tasks. Activations are captured from the action expert's residual stream at layers $\{0, 5, 11, 17\}$ during every inference call.

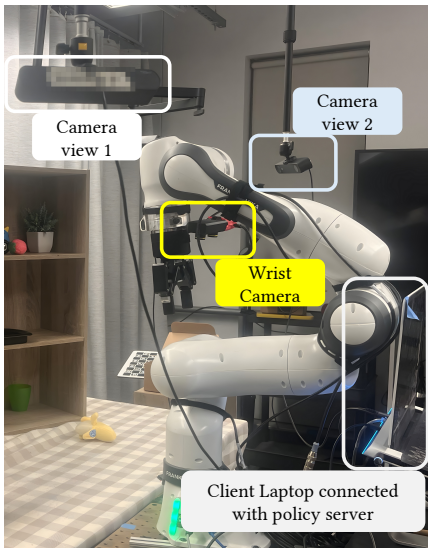


Figure 6: **Hardware configuration for real-world evaluation.** We use the DROID (Khazatsky et al., 2024) setup: a 7-DoF Franka Emika Panda arm, a Robotiq 2F-85 parallel-jaw gripper, a wrist-mounted ZED Mini RGB-D camera, and two side-mounted ZED 2 stereo cameras.

Per-task calibration data and hyperparameters. For each task, the 15 base-policy trials serve both as the baseline measurement and as the fitting rollouts for COAST; no additional data collection is required. Table 4 lists the language prompt, episode horizon, and the selected steering layer, aperture α, and strength β for every task, and the success/failure split of the fitting rollouts.

Results with confidence intervals. Table 5 reports, for each task, the number of successful trials out of 15 for the base policy and for COAST, together with Wilson 95% confidence intervals. The

Table 3: Real-robot evaluation protocol.

Parameter	Value
Platform	DROID: 7-DoF Franka Emika Panda, Robotiq 2F-85 gripper, wrist ZED Mini and external ZED 2
Policy	$\pi_{0.5}$-DROID, not fine-tuned on any demonstrations of these tasks
Control	Joint velocity and gripper position at 15 Hz
Action chunk	10 actions $\times$ 8 dims (7 joint velocities + 1 gripper)
Open-loop horizon	8 of 10 actions executed per replan (about 0.53 s)
Cameras to policy	1 external ZED (left) + wrist, resized with padding to 224×224
Activation capture	Residual stream at layers $\{0, 5, 11, 17\}$; tensor of (10 denoising steps, 4 layers, 15 action tokens, d=1024) per inference call
Trials	15 evaluation trials per task and condition; the 15 base-policy trials are also the fitting rollouts
Scene	Target-object placements kept as consistent as possible across trials and conditions

Table 4: Per-task prompts, horizons, and selected COAST hyperparameters on the real robot.

Task	Layer	α	β	Prompt	Max timesteps
Open Drawer	5	5.0	0.2	"open upper left drawer"	700
Close Microwave	5	3.5	0.1	"close the microwave door tightly"	500
Put Duck in Cabinet	5	0.2	0.1	"put duck on cabinet"	200

mean absolute gain over the three tasks is $+0.40$. On `Close Microwave`, the success conceptor is fit from a single successful base-policy episode (App. F.3).

Table 5: Real-robot success rates with Wilson 95% confidence intervals (N=15 trials per task and condition). "Fitting S/F" is the success/failure split of the 15 base-policy rollouts used to fit the conceptor.

Task	Fitting S/F	Base	+COAST	Gain
Open Drawer	5/10	0.33 (5/15) [0.15, 0.58]	0.86 (13/15) [0.62, 0.96]	$+0.53$
Close Microwave	1/14	0.07 (1/15) [0.01, 0.30]	0.46 (7/15) [0.25, 0.70]	$+0.39$
Put Duck in Cabinet	2/13	0.13 (2/15) [0.04, 0.38]	0.40 (6/15) [0.20, 0.64]	$+0.27$

Scope of the real-world evidence. This evaluation is deliberately reported with its limitations. It covers three tasks with 15 trials per condition in a single physical setup, and the confidence intervals are correspondingly wide: the base and COAST intervals are disjoint only for `Open Drawer`. The results should therefore be read as evidence that the simulation gains transfer to hardware on the evaluated tasks, not as an estimate of robustness across varied real-world scenes. All 15 COAST evaluation rollouts per task and the base-policy activations used for fitting are provided through an anonymized link in the supplementary material. Example rollouts are shown in Figure 7.

B Stronger Fine-Tuning Baseline and Matched Tuning Protocol for Steering Baselines

B.1 SFT with 200 trajectories and up to 1,000 steps

The SFT baseline in Table 1 uses 30 on-policy rollouts per task and 200 LoRA steps (App. N.1). To test whether its underperformance is an artifact of a starved budget, we scale the same filtered-BC recipe to 200 on-policy rollouts per task for $\pi_{0.5}$ on MetaWorld and train for up to 1,000 steps, evaluating merged checkpoints at 200, 500, 800, and 1,000 steps on 30 held-out test rollouts per cell. Successful episodes among the 200 rollouts become training samples. "Base" is the pooled base-policy success rate over the 200 collection rollouts and therefore differs from the 30-rollout test-set base in Table 1.

Two findings stand out. First, more data and more steps do not close the gap: the best step count (1,000) still averages 2.7 points below the base policy, and performance is non-monotonic across

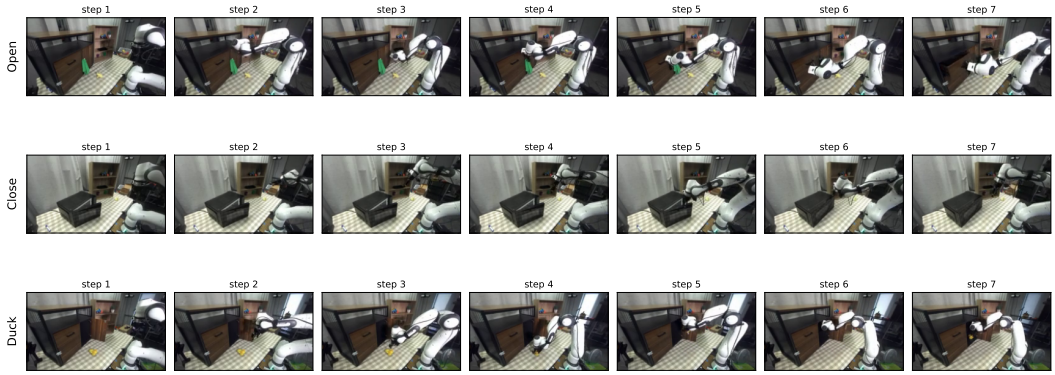


Figure 7: **Successful Real World Rollouts of $\pi_{0.5}$-COAST.** Example COAST trajectories on `Open Drawer`, `Close Microwave`, and `Put Duck in Cabinet` (15 independent trials per task and condition).

Table 6: Filtered-BC SFT with 200 on-policy rollouts per task, $\pi_{0.5}$ on MetaWorld ML45 (success rate in %, 30 held-out test rollouts per cell). Δ compares the 1,000-step checkpoint with the base policy.

Task	**Base**	**200**	**500**	**800**	**1,000**	Δ
coffee-push	77.5	90	80	93	90	$+13$
faucet-close	78.5	90	70	80	90	$+12$
disassemble	54.0	53	13	53	63	$+9$
reach	79.0	77	77	80	83	$+4$
pick-place	69.5	90	70	67	73	$+4$
push	87.5	87	80	87	87	0
stick-push	25.5	30	20	27	20	-6
coffee-pull	91.0	83	90	70	77	-14
plate-slide-back	54.0	3	13	37	43	-11
pick-place-wall	39.5	10	13	10	3	-37
Mean	65.6	61.3	52.6	60.4	62.9	-2.7

steps (52.6% at step 500), indicating instability of the merged adapters at this data scale. Second, SFT regresses exactly where COAST gains most. SFT's largest regression is `pick-place-wall` ($39.5 \rightarrow 3$), the same task on which COAST produces its largest gain from only 15 rollouts ($0.20 \rightarrow 0.87$ under global steering on held-out rollouts, Table 1). Over the same ten tasks, COAST gains $+0.25$ from 15 rollouts.

We stress the scope of this comparison. It shows that COAST outperforms the evaluated LoRA-based filtered-BC recipe at two data budgets; it does not show that COAST is better than every way of updating the policy through training. We do not compare against other fine-tuning recipes or reinforcement-learning post-training, and claims about gradient-based adaptation in this paper should be read as relative to the evaluated SFT baseline.

B.2 CAA AND SAE UNDER COAST'S TRAIN/TEST PROTOCOL

In Table 1, the CAA and SAE baselines selected their steering strength post hoc on the 30 held-out test rollouts (App. N.2, App. N.3), whereas COAST selected its configuration on the 15 fitting rollouts only. This asymmetry favors the baselines. To remove it, we re-ran CAA and SAE under COAST's exact train/test protocol with wider hyperparameter grids, so that every method evaluates the same number of configurations per task:

- CAA: $\alpha \in \{0.1, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 3.0, 5.0\}$ (9 configurations per task);
- SAE: $\alpha \in \{0.1, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 3.0, 4.0\}$ (9 configurations per task);
- COAST: aperture $\alpha \in \{0.1, 0.5, 1.0\} \times$ strength $\beta \in \{0.1, 0.3, 0.5\}$ (9 configurations per task).

All methods use the identical 15 base-policy rollouts for activation collection and hyperparameter selection, and are evaluated on the same 30 held-out rollouts.

Table 7: Mean success rate with CAA and SAE tuned under COAST's train/test protocol (9 configurations per task per method). Under the matched protocol COAST's margin over both baselines grows relative to Table 1, where CAA and SAE were tuned post hoc on the test set.

Suite	Base	+Glob.	+Per.	+CAA (matched)	+SAE (matched)
$\pi_{0.5}$ LIBERO-10	0.43	0.76	**0.80**	0.45	0.60
$\pi_{0.5}$ RoboCasa	0.40	**0.55**	**0.55**	0.40	0.45
GR00T N1.5 RoboCasa	0.59	**0.75**	0.71	0.60	0.65

Table 1 retains the original, post-hoc-tuned CAA and SAE numbers, which are an upper bound for those baselines; the matched-protocol numbers in Table 7 are the like-for-like comparison.

B.3 SFT ROLLOUT BUDGET

SFT uses $N{=}30$ on-policy rollouts per task, twice the 15 rollouts available to COAST. Following the filtered-BC recipe, only action chunks from successful episodes are kept (App. N.1.1), so the effective training set is the success subset of those 30 rollouts. The larger rollout budget and the per-task LoRA fine-tune and merge give SFT a strict per-task advantage over the steering intervention. SFT's underperformance in Table 1 therefore cannot be attributed to a starved baseline.

C DOES COAST ONLY HELP UNDERTRAINED POLICIES?

The largest absolute gains in Table 1 (MetaWorld and LIBERO) are obtained on early checkpoints chosen for mixed success/failure outcomes, whereas the fully trained RoboCasa checkpoints show gains of $+0.14$ to $+0.16$. A natural reading is that COAST mainly recovers undertrained competence. Three analyses test this reading.

C.1 HEADROOM ANALYSIS

We stratify all 60 task–model cells of Table 1 by baseline success rate and use the global contrastive column uniformly, with no per-task strategy selection. "Headroom recovered" is the gain divided by $(1 - \text{Base})$.

Table 8: Gain and fraction of headroom recovered by COAST (global) as a function of baseline success rate, over the 60 task–model cells of Table 1.

Baseline stratum	Tasks	Base	+COAST (Glob.)	Gain	Headroom recovered
Base > 0.70	16	0.82	0.93	$+0.10$	59%
$0.40 \leq$ Base ≤ 0.70	29	0.58	0.81	$+0.22$	54%
Base < 0.40	15	0.20	0.45	$+0.26$	32%

Absolute gains shrink as baselines rise because the available headroom shrinks. The fraction of headroom recovered, however, rises from 32% on low-baseline tasks to 59% on high-baseline tasks. If COAST only recovered undertrained competence, this fraction would fall on strong tasks; it does the opposite.

C.2 FULLY TRAINED CHECKPOINTS

All three RoboCasa checkpoints are fully trained production checkpoints (App. M). COAST adds $+0.14$ to $+0.16$ on each, all significant under the pooled z-test (Table 9). On the GR00T N1.5 tasks whose baseline is already high (base ≥ 0.67: `Close Fridge`, `PP Cabinet`, `PP Stove`, `Kettle`), COAST adds $+0.15$ on average with zero regressions.

We note that a fully trained checkpoint is not necessarily a near-state-of-the-art policy: the evaluated RoboCasa checkpoints reach 32–59% mean success, and COAST's value on a policy that already succeeds on most tasks is tested here only through the high-baseline stratum of Table 8 and the

Table 9: COAST on fully trained RoboCasa checkpoints (global strategy for $\pi_{0.5}$ and GR00T N1.5; per-step for Diffusion Policy).

Fully trained checkpoint	**Base**	**+COAST**	**Gain**	**Significance**
$\pi_{0.5}$ RoboCasa (75k steps)	0.40	0.56	+0.16	$p_z = .009$
GR00T N1.5 RoboCasa (120k steps)	0.59	0.75	+0.16	$p_z = .006$
Diffusion Policy RoboCasa (epoch 1500)	0.32	0.46	+0.14	$p_z = .035$

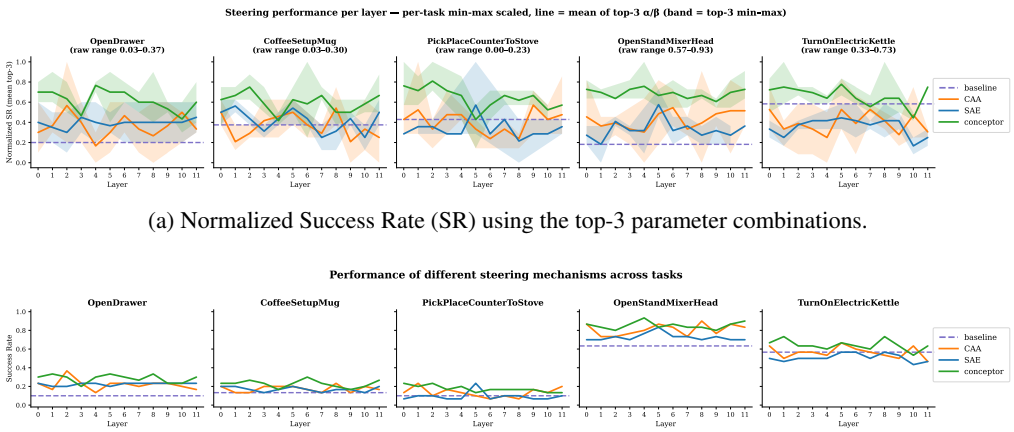
RoboCasa

Task	**300**		**600**		**900**		**1200**		**1500**	
	Base	Best	Base	Best	Base	Best	Base	Best	Base	Best
CloseFridge	0.57	**0.53** (global_L5_a2.0_b0.3)	0.27	**0.30** (per_step_L5_a2.0_b0.1)	0.40	**0.47** (global_L5_a2.0_b0.1)	0.40	**0.43** (global_L5_a2.0_b0.1)	0.43	**0.60** (per_step_ds0_L5_a2.0_b0.1)
CoffeeSetupMug	0.00	–	0.03	**0.03** (global_L5_a0.1_b0.1)	0.03	**0.07** (global_L5_a0.1_b0.1)	0.13	**0.20** (global_L5_a2.0_b0.1)	0.13	**0.23** (global_L5_a1.0_b0.1)
OpenDrawer	0.07	**0.17** (global_L5_a0.1_b0.3)	0.03	**0.17** (global_L11_a0.1_b0.3)	0.03	**0.10** (per_step_ds0_L8_a0.1_b0.3)	0.07	**0.20** (global_L5_a0.1_b0.3)	0.10	**0.33** (global_L5_a0.1_b0.3)
OpenStandMixerHead	0.60	**0.83** (per_step_ds0_L5_a2.0_b0.1)	0.63	**0.87** (global_L5_a2.0_b0.1)	0.57	**0.80** (global_L5_a2.0_b0.1)	0.53	**0.87** (per_step_ds0_L5_a2.0_b0.1)	0.63	**0.87** (linear_L5_la1.0)
PickPlaceCounterToCabinet	0.20	**0.30** (global_L5_a2.0_b0.1)	0.13	**0.23** (global_L5_a1.0_b0.3)	0.20	**0.40** (global_L5_a2.0_b0.3)	0.30	**0.53** (global_L5_a1.0_b0.3)	0.30	**0.57** (per_step_ds0_L5_a2.0_b0.3)
PickPlaceCounterToStove	0.13	**0.10** (global_L5_a0.5_b0.1)	0.07	**0.10** (global_L5_a0.5_b0.1)	0.10	**0.07** (global_L5_a0.5_b0.1)	0.13	**0.13** (global_L5_a0.5_b0.1)	0.10	**0.13** (global_L5_a0.5_b0.1)
TurnOnElectricKettle	0.37	**0.50** (per_step_ds0_L5_a10.0_b0.3)	0.40	**0.63** (per_step_ds0_L5_a2.0_b0.3)	0.33	**0.57** (per_step_ds0_L5_a2.0_b0.3)	0.60	**0.67** (per_step_ds0_L5_a10.0_b0.3)	0.57	**0.67** (per_step_ds0_L5_a10.0_b0.1)

Table 10: **Task performance across training steps.** We evaluate the base vs. best configuration at epochs 300, 600, 900, 1200, and 1500.

positive-only ablation on tasks with baseline ≥ 0.67 (App. F.6). RoboCasa is also a harder benchmark than MetaWorld and LIBERO-10 (randomized kitchen layouts, objects, textures, and lighting), so its absolute gains are not directly comparable with those on the easier suites.

C.3 Diffusion Policy epoch sweep and layer robustness



(a) Normalized Success Rate (SR) using the top-3 parameter combinations.

(b) Absolute peak Success Rate achieved across all combinations.

Figure 8: **Steering Performance by Layer. (a)** Normalized SR using the top-3 parameter combinations per method at each layer. The solid line represents the mean of the top-3 configurations, while the shaded band shows the spread (min-max). COAST was evaluated over 12 combinations (α, β, strategy), CAA over 3 (α), and SAE over 2 (α). **(b)** Absolute peak Success Rate achieved at each layer. COAST consistently outperforms both the unsteered baseline and the CAA/SAE steering methods across different tasks and layers, demonstrating low variance and high robustness to layer selection.

A natural concern is that the gains from steering could depend on a favorable checkpoint, layer, or hyperparameter choice rather than reflecting a robust, generalized effect. To rule this out, we evaluate our steering mechanism across multiple stages of training, intervention layers, and parameter grids, comparing COAST against both its unsteered base policy and internal baselines under the same evaluation protocol.

Setup. We consider five Diffusion Policy checkpoints saved at epochs 300, 600, 900, 1200, and 1500. For each checkpoint, we run the unsteered base policy on the 7 RoboCasa tasks and separately report the best steered result obtained from the same checkpoint under our steering sweep. This produces a per-epoch comparison between the base policy and its strongest steered variant. Im-

portantly, the checkpoint set is fixed in advance and spans the training trajectory uniformly, ensuring the analysis does not rely on post hoc checkpoint cherry-picking.

Additionally, to evaluate robustness across network depth, we swept over a grid of parameters for each method at every layer (0 to 11) for epoch 1500. For Contrastive Activation Addition (CAA), we evaluated $\alpha \in \{0.1, 0.5, 1.0\}$. For Sparse Autoencoder (SAE) steering, we evaluated $\alpha \in \{0.5, 1.0, 2.0\}$. For COAST, we swept aperture $\alpha \in \{0.5, 1.0, 2.0\}$, steering strength $\beta \in \{0.1, 0.3\}$, and strategy $\in \{\text{global}, \text{per-step}\}$, resulting in 12 candidate combinations per layer.

Aggregating all 7 RoboCasa tasks (n=210 rollouts per cell) at each checkpoint gives Table 11; the per-task breakdown is in Table 10.

Table 11: Diffusion Policy on RoboCasa across training epochs, aggregated over 7 tasks (n=210 rollouts per cell). "Improved / regressed" counts tasks whose steered success rate is above / below the base checkpoint.

Epoch	**300**	**600**	**900**	**1200**	**1500 (converged)**
Base	0.28	0.22	0.24	0.31	0.32
+COAST	0.35	0.33	0.35	0.43	0.49
Gain	+0.07	+0.11	+0.12	+0.12	+0.16
Tasks improved / regressed	4 / 2	6 / 0	6 / 1	6 / 0	7 / 0

Three observations follow. (1) The gain grows monotonically with training, from $+0.07$ at epoch 300 to $+0.16$ at epoch 1500; the largest absolute gain occurs at the most converged checkpoint, not the least. (2) Regressions vanish as the policy converges: at epoch 300 two tasks degrade under steering, whereas at epoch 1500 all 7 tasks improve with zero regressions (one-sided sign test, $p = .008$). Steering is more reliable on a better-trained policy, because the behaviors it modulates are better formed. (3) Steering and training compose rather than substitute. Base performance is nearly flat from epoch 300 to 1500 ($+0.046$), while steering at epoch 300 already yields $+0.07$; the steered epoch-300 policy (0.35) outperforms the unsteered fully trained epoch-1500 policy (0.32), and the combination (0.49) exceeds either alone. Continuing to train these checkpoints is therefore not a substitute for steering.

Results. Table 10 shows that the improvement from steering is not isolated to one stage of training. Across all evaluated epochs, the steered policy consistently matches or improves upon the corresponding base checkpoint. This is especially important because the absolute base performance varies substantially with the epoch, yet the relative advantage of steering remains present across the entire sweep.

Furthermore, COAST demonstrates strong performance robustness across different model layers. As shown in Figure 8a, plotting the mean and variance of the top-3 parameter combinations at each layer reveals that COAST not only achieves a higher normalized success rate than CAA and SAE across almost all layers, but also maintains a highly stable performance band. Figure 8b visualizes the absolute maximum success rate achieved at each layer. Across five distinct RoboCasa tasks, COAST consistently outperforms the baselines, and the performance variance between adjacent layers is relatively small.

Discussion. These results strengthen the claim that the steering effect is genuine and broadly applicable. Because every checkpoint is evaluated independently against its own base model, the gains shown across early, middle, and late checkpoints confirm that the intervention is compatible with the learned policy throughout training. Similarly, the layer and hyperparameter analysis indicates that conceptor-based steering is not overly brittle or sensitive to the exact layer of intervention, provided our efficient hyperparameter selection heuristic (Sec. I.2) is applied. Together, these findings demonstrate that COAST provides holistic, resilient improvements independent of specific model configurations.

D CROSS-TASK TRANSFER WITHOUT ORACLE SOURCE SELECTION

Table 2 reports, for each target task, the best-performing source conceptors, which is an oracle choice. This appendix (i) states precisely what data an oracle-free selection rule needs and (ii) reports transfer with no source selection at all.

D.1 WHAT AN ORACLE-FREE SELECTION RULE REQUIRES

Failure-subspace containment $\mathrm{cf}(\mathrm{src} \to \mathrm{tgt}) = \mathrm{tr}(C^f_{\mathrm{src}} C^f_{\mathrm{tgt}})/\mathrm{tr}((C^f_{\mathrm{src}})^2)$ (Sec. 4.4) requires the target task's *failure* conceptor, i.e., activations from unsteered target rollouts that failed, but it requires neither target successes nor any steered rollouts. The setting we have in mind is therefore one in which a batch of unsteered target-task rollouts already exists, possibly consisting entirely of failures (in which case the target's own contrastive conceptor cannot be fit), and no new rollouts are collected for steering. Containment is significantly correlated with the realized transfer gain (Pearson r=0.30, p=4.5e−3 on LIBERO-10; r=0.49, p=1.1e−3 on RoboCasa), but the correlation is moderate: it ranks candidate sources, it does not guarantee that the top-ranked source transfers well.

D.2 AVERAGE-SOURCE TRANSFER WITH NO SELECTION

As a selection-free alternative, we steer every target task with the uniform average of all other tasks' contrastive conceptors (same benchmark, same layer and strategy), so that no target data of any kind is used to choose the operator. Tables 12 and 13 report the results for $\pi_{0.5}$ on LIBERO-10 and RoboCasa. Base success rates in these tables are measured on the rollouts used for the transfer experiments and differ slightly from Table 1.

Table 12: Average-source transfer, $\pi_{0.5}$ LIBERO-10.

Target	Base	Averaged
Stove+Moka (KS3)	0.67	0.67
Bowl+Drawer (KS4)	0.60	0.60
Mug+Micro (KS6)	0.33	0.40
Two Mokas (KS8)	0.00	0.33
Soup+Cheese (LR1)	0.60	0.80
Soup+Tomato (LR2a)	0.33	0.80
Cheese+Butter (LR2b)	0.67	0.87
Mugs+Plates (LR5)	0.07	0.47
Mug+Choc (LR6)	0.87	0.87
Book+Caddy (ST1)	0.80	0.93
Mean	0.49	**0.67**

Table 13: Average-source transfer, $\pi_{0.5}$ RoboCasa.

Target	Base	Averaged
Close Fridge	0.20	0.33
Coffee Mug	0.13	0.40
Open Drawer	0.53	0.40
Stand Mixer	0.67	0.67
PP Cabinet	0.60	0.60
PP Stove	0.33	0.53
Kettle	0.33	0.33
Mean	0.40	**0.47**

The averaged conceptor improves the mean success rate on both suites ($+0.18$ on LIBERO-10, $+0.07$ on RoboCasa) without any source selection. Averaging pools the failure-suppression directions of all source tasks; since failure subspaces overlap substantially across tasks while success directions are task-specific (Figure 5), the averaged operator primarily suppresses shared failure structure. Averaging is not uniformly safe, however: it reduces performance on `Open Drawer` ($0.53 \to 0.40$) and leaves several tasks unchanged.

Framing. Standard use of COAST requires 15 target-task rollouts with success/failure labels and a small hyperparameter check (App. F). Transfer of previously fitted conceptors removes the need for new target rollouts, but selecting a good source still benefits from target failure data, and the selection-free average is a weaker operator than the oracle choice. We therefore describe this regime as *cross-task transfer without new target rollouts* rather than zero-shot transfer, and regard it as a preliminary result.

E STATISTICAL ROBUSTNESS AND MULTIPLE-COMPARISON CORRECTION

Primary test. Table 1 reports two significance tests. The pooled two-proportion z-test (p_z) treats episodes as independent observations and can be inflated by within-task correlation of episodes.

The paired t-test across tasks (p_t) treats each task's mean success rate as a single observation, so within-task episode correlation cannot inflate it; we regard p_t as the primary test. All 14 contrastive comparisons (global and per-step across the seven model–benchmark pairs) pass $p_t < .05$. Under the pooled z-test, 13 of 14 pass, with Diffusion Policy global at $p_z = .058$.

Multiple comparisons. We treat each model–benchmark pair as a separate family, since each answers one question ("does COAST help this policy on this benchmark?"), and within each family we correct over the three COAST strategies (global, per-step, positive-only; m=3), which are variants of the same method rather than independent hypotheses. Table 14 lists the Bonferroni- and Holm–Bonferroni-adjusted values of p_t. Under Holm–Bonferroni, all 14 contrastive results remain significant; under the more conservative plain Bonferroni correction, 12 of 14 remain significant (the exceptions are per-step steering for GR00T N1.5 on RoboCasa and for π_0-FAST on MetaWorld, whose global counterparts remain significant).

Table 14: Paired t-test p-values from Table 1 with Bonferroni (Bonf.) and Holm–Bonferroni (Holm) adjustment within each model–benchmark family (m=3 strategies). Values above .05 are marked n.s.

	Global			**Per-step**			**Positive-only**		
Model–benchmark	p_t	Bonf.	Holm	p_t	Bonf.	Holm	p_t	Bonf.	Holm
$\pi_{0.5}$ RoboCasa	.002	.006	.006	.009	.027	.018	.044	.132 (n.s.)	.044
GR00T N1.5 RoboCasa	.002	.006	.006	.039	.117 (n.s.)	.039	.011	.033	.022
Diffusion Policy RoboCasa	.004	.012	.012	.006	.018	.012	.123	.369 (n.s.)	.123 (n.s.)
$\pi_{0.5}$ LIBERO-10	$< .001$	$< .003$	$< .003$	$< .001$	$< .003$	$< .003$	.001	.003	.003
π_0-FAST LIBERO-10	$< .001$	$< .003$	$< .003$	$< .001$	$< .003$	$< .003$	.004	.012	.004
$\pi_{0.5}$ MetaWorld ML45	.008	.024	.018	.012	.036	.018	.006	.018	.018
π_0-FAST MetaWorld ML45	.003	.009	.009	.023	.069 (n.s.)	.036	.018	.054 (n.s.)	.036

Sample sizes. All simulation results use 15 rollouts per task for fitting and hyperparameter selection and 30 held-out rollouts per task and steering condition for testing (App. K.5). The real-robot evaluation uses 15 trials per task and condition because of physical cost; App. A reports Wilson 95% confidence intervals for every real-robot cell.

F Calibration Cost, Relaxed Data Regimes, and Eligibility for Contrastive Fitting

F.1 Calibration cost

COAST avoids weight updates, but its standard contrastive form does require per-task calibration. The complete cost is: (i) 15 uncurated base-policy rollouts on the target task with binary success labels (these are ordinary evaluation rollouts, not demonstrations); (ii) a closed-form conceptor fit that takes under one second per (task, layer, aperture) on a CPU (App. L); and (iii) a check of two values of the steering strength β once the layer and aperture have been chosen geometrically (App. I.2). Inference overhead is equal to that of any hook-based additive steering method (App. P). Table 15 shows that this budget is sufficient even when the base policy rarely succeeds.

Table 15: Calibration data and outcome for representative $\pi_{0.5}$ tasks. "Fitting S/F" is the success/failure split among the 15 fitting rollouts; success rates are on 30 held-out test rollouts (best contrastive strategy from Table 1).

Task	**Suite**	**Fitting S/F (of 15)**	**Base**	**+COAST**	**Gain**
Mugs+Plates (LR5)	LIBERO-10	1 / 14	0.07	0.60	+0.53
Mug+Micro (KS6)	LIBERO-10	2 / 13	0.13	0.60	+0.47
Two Mokas (KS8)	LIBERO-10	3 / 12	0.20	0.53	+0.33
Close Fridge	RoboCasa	2 / 13	0.20	0.47	+0.27
Kettle	RoboCasa	2 / 13	0.33	0.53	+0.20
Soup+Cheese (LR1)	LIBERO-10	12 / 3	0.80	1.00	+0.20

F.2 RELAXED DATA REGIMES

Paired per-task success *and* failure rollouts are not strictly required. Table 16 summarizes four regimes evaluated in this paper that relax the data requirement.

Table 16: Data regimes that relax the requirement for paired success/failure rollouts.

Regime	**Success rollouts**	**Failure rollouts**	**Result**
Positive-only (Sec. 3.2; App. F.6)	yes	none	$+0.07$ to $+0.20$ mean gain on all seven model–benchmark pairs (Table 1); never regresses on high-baseline tasks
Failure-only (App. F.7)	none	yes	3 of 4 zero-success tasks lift off 0 (`dial-turn`: $0.00 \rightarrow 0.38$)
Single success (App. F.3)	1 episode	yes	LR5: $0.07 \rightarrow 0.60$; `Close Microwave` on hardware: $0.07 \rightarrow 0.46$
Transfer of previously fitted conceptors (Sec. 4.4; App. D)	none new	none new	Transferred conceptors retain most of the self-fit gain; selection-free averaging gives $+0.18$ (LIBERO-10) and $+0.07$ (RoboCasa)

F.3 ELIGIBILITY CRITERIA AND THE FITTING UNIT

The unit of fitting is the activation vector, not the episode. Each inference call yields one mean-pooled residual vector per denoising step (10 steps for $\pi_{0.5}$; App. J), and a LIBERO-10 episode contains roughly 25–30 replan steps, so a single successful episode contributes several hundred vectors to C^+. This is how the success conceptor for LR5 (1 success in 15 fitting rollouts) and for `Close Microwave` on the real robot (1 success in 15) is fit.

Two episode-level eligibility criteria were applied and are stated here explicitly. For π_0-FAST we required at least three episodes per class; this is why `Cheese+Butter` (LR2b), which had zero failures under the π_0-FAST checkpoint, is excluded from the contrastive columns of Table 1. For $\pi_{0.5}$ on LIBERO-10 and RoboCasa we required at least one episode per class, relying on the activation-level sample count; the default value of three per class in our code was overridden for these runs. Tasks with three or fewer successful fitting episodes are marked with ‡ in Table 1. These thresholds only control which tasks are eligible for contrastive fitting; every reported result is evaluated on 30 held-out test rollouts with disjoint seeds (App. K.5), so the thresholds do not affect the validity of the reported success rates.

F.4 ACTIVATION DATASET COMPOSITION

For each model-benchmark cell of Table 1, we fit conceptors and SAE steering vectors from a single N-episode rollout of the corresponding policy checkpoint. Success and failure labels come from each simulator's task-completion criterion, recorded as `episode_success` in each episode's `metadata.json`. Tables 17 through 19 report per-task success and failure counts.

The contrastive fit requires very few episodes per class to produce a usable $C_{\text{contrastive}}$ or $v_{\text{contrastive}}$ (see the eligibility criteria in App. F.3: at least three episodes per class for π_0-FAST and at least one episode per class for $\pi_{0.5}$, where each episode contributes hundreds of per-step activation vectors). This low bar is what makes the recipe practical at the checkpoint scales we evaluate. Many tasks at our $\pi_{0.5}$ checkpoints (LIBERO t=2,000, RoboCasa t=75,000) succeed on only 1 to 3 of 15 rollouts, and those handful of success trajectories are exactly what the contrastive operation needs. App. F.5 characterizes this low-success regime quantitatively.

F.5 STEERING GAINS UNDER LOW-SUCCESS REGIMES

A practical concern with any contrastive recipe is whether enough success trajectories exist to fit a usable contrastive direction. At our $\pi_{0.5}$ checkpoints this concern is acute. Of the 17 LIBERO-10 and RoboCasa tasks under $\pi_{0.5}$ in Tables 17 and 19, six have at most three successful episodes ($S \leq 3$), and another four have $S \leq 6$. Over half of these tasks therefore have three or fewer successful fitting episodes.

Table 17: LIBERO-10 activation dataset composition. N=15 episodes per task; cells report success / failure counts.

Task	$\boldsymbol{\pi_{0.5}}$ (S/F)	$\boldsymbol{\pi_0}$-FAST (S/F)
KS3	8 / 7	12 / 3
KS4	6 / 9	10 / 5
KS6	2 / 13	7 / 8
KS8	3 / 12	5 / 10
LR1	12 / 3	10 / 5
LR2a	6 / 9	10 / 5
LR2b	9 / 6	15 / 0
LR5	1 / 14	9 / 6
LR6	8 / 7	11 / 4
ST1	10 / 5	9 / 6
Total	65 / 85 ($\overline{\text{SR}}$=.43)	98 / 52 ($\overline{\text{SR}}$=.65)

Table 18: MetaWorld ML45 activation dataset composition for the 10 paper-table tasks. N=16 episodes per task (16 parallel envs $\times$ 1 episode each).

Task	$\boldsymbol{\pi_0}$-FAST (S/F)
coffee-pull-v3	10 / 6
coffee-push-v3	14 / 2
disassemble-v3	9 / 7
faucet-close-v3	7 / 9
pick-place-v3	11 / 5
pick-place-wall-v3	11 / 5
plate-slide-back-v3	14 / 2
push-v3	12 / 4
reach-v3	13 / 3
stick-push-v3	13 / 3
Total	114 / 46 ($\overline{\text{SR}}$=.71)

These same low-success tasks produce the largest absolute gains from steering in Table 1; Table 15 lists the fitting split, baseline, and steered success rate for representative tasks.

The pattern is consistent across both benchmarks and across all three conceptor variants. Two factors plausibly contribute. First, the contrastive operation $C_s \cdot \neg C_f$ remains well-defined when S is small, because the failure subspace is estimated from $n \geq 12$ trajectories and contributes most of the discriminative geometry. Second, low-baseline tasks leave more absolute headroom for steering to recover.

The complementary observation appears in the high-baseline regime. On MetaWorld under π_0-FAST ($\overline{\text{SR}}$=.71) and on the success-rich RoboCasa GR00T tasks such as `Close Fridge` (.80), absolute gains shrink toward zero or saturate near the ceiling. Taken together, these results indicate that a handful of success trajectories on a struggling checkpoint suffice for the contrastive recipe. We stress that the shrinking *absolute* gains on high-baseline tasks reflect headroom arithmetic (success rate is capped at 1.0), not a loss of effectiveness: App. C.1 shows that the *fraction* of remaining headroom recovered by COAST is in fact highest on high-baseline tasks.

F.6 Positive-Only Conceptor Steering on High-Baseline Tasks

A natural concern is that conceptor steering might only help on tasks with mixed outcomes, where a contrastive conceptor $C_{\text{succ}} \wedge \neg C_{\text{fail}}$ can separate successes from failures. On tasks where the base policy already succeeds every rollout, there is no failure set to contrast against. We therefore test whether the *positive-only* conceptor C_{succ} can be applied safely to such tasks without eroding their success rate.

Setup. We run $\pi_{0.5}$ on the 18 MetaWorld-ML45 tasks whose unsteered baseline is ≥ 0.67 over 15 rollouts. Conceptors are fit on layer-11 activations from the pre-collected ML45 activation dataset (anonymized link in the supplementary material). We sweep aperture $\alpha \in \{0.1, 0.5, 1.0\}$ at fixed

Table 19: RoboCasa activation dataset composition. N=15 episodes per task; cells report success / failure counts.

Task	$\boldsymbol{\pi_{0.5}}$ (S/F)	**GR00T N1.5** (S/F)
Close Fridge	2 / 13	12 / 3
Coffee Mug	4 / 11	3 / 10
Open Drawer	9 / 6	8 / 7
Stand Mixer	10 / 5	11 / 4
PP Cabinet	7 / 8	11 / 4
PP Stove	7 / 8	9 / 6
Kettle	2 / 13	7 / 8
Total	41 / 64 ($\overline{\text{SR}}$=.39)	61 / 42 ($\overline{\text{SR}}$=.59)

$\beta = 0.2$, and compare a single *global* conceptor against a *per-step* conceptor tied to each of the 10 denoising steps (6 configurations per task). Each is evaluated over 30 rollouts with a 300-step horizon. As a control, we additionally run a matched-quota *random* conceptor that projects onto a random subspace of the same trace as C_{succ}.

Results. Table 20 reports, per task, the baseline success rate and the best rate across the six positive-only configurations. Positive-only steering *never regresses* below the baseline and strictly improves two tasks that were not already at ceiling: FAUCET-OPEN-V3 ($0.93 \rightarrow 1.00$) and PEG-UNPLUG-SIDE-V3 ($0.67 \rightarrow 1.00$).

Task	Baseline	Best Pos-Only	Best Configuration
button-press-topdown-v3	1.00	1.00	global, α=0.1
button-press-topdown-wall-v3	1.00	1.00	global, α=0.1
button-press-v3	1.00	1.00	global, α=0.1
button-press-wall-v3	1.00	1.00	global, α=0.1
coffee-button-v3	1.00	1.00	global, α=0.1
door-close-v3	1.00	1.00	global, α=0.1
drawer-close-v3	1.00	1.00	global, α=0.1
drawer-open-v3	1.00	1.00	global, α=0.1
faucet-open-v3	0.93	**1.00**	global, α=0.1
handle-press-side-v3	1.00	1.00	per-step, α=0.1
handle-press-v3	1.00	1.00	global, α=0.1
peg-unplug-side-v3	0.67	**1.00**	per-step, α=1.0
plate-slide-side-v3	1.00	1.00	global, α=0.1
plate-slide-v3	1.00	1.00	global, α=0.1
push-wall-v3	1.00	1.00	per-step, α=1.0
reach-wall-v3	1.00	1.00	global, α=0.5
window-close-v3	1.00	1.00	global, α=0.1
window-open-v3	1.00	1.00	global, α=0.1
Mean	**0.98**	**1.00**	—

Table 20: Positive-only conceptor steering on 18 high-baseline MetaWorld-ML45 tasks. Success rates are over 15 rollouts. "Best Pos-Only" is the maximum across six configurations ($\alpha \in \{0.1, 0.5, 1.0\}$, $\beta = 0.2$, global vs. per-step). Bold entries mark strict improvements over baseline.

Discussion. Two patterns stand out. The most frequently selected configuration is the gentlest — a single global conceptor at $\alpha = 0.1$ — which matches the intuition that when activations already lie inside the success manifold, only a light projection is needed. The two tasks where stronger or temporally-resolved steering helped are precisely the two whose baselines were below ceiling, suggesting the per-step conceptor is most useful when the policy has genuine failure modes to correct at specific denoising steps. Overall, positive-only conceptor steering behaves as a safe default: a no-op on saturated tasks and a rescue operator on borderline ones.

F.7 Failure-only ablation: does suppressing failure directions alone recover success?

The contrastive conceptor $C^{\text{con}} = C^s \wedge \neg C^f$ combines a positive projection onto success directions with a negation of failure directions. A natural question is how much of the observed gain comes from each component. We isolate the failure-suppression term by steering with $\neg C^f$ alone, fit on the failure trajectories from the π_0-FAST MetaWorld activation rollouts, on four tasks where the baseline checkpoint succeeds zero out of sixteen episodes. With no success examples in the dataset, C^s cannot be fit, so any gain in this setting is attributable entirely to suppression of failure geometry.

Table 21: Failure-only steering with $\neg C^f$ on π_0-FAST MetaWorld tasks at zero baseline. Each row reports the best success rate across an (α, β) sweep at L=11, with the winning condition listed. Three of four tasks recover nonzero success without ever observing a successful trajectory.

Task	**Baseline**	**Best $\neg C^f$**	**Condition**
`assembly-v3`	0%	0%	—
`dial-turn-v3`	0%	38%	$\alpha \in \{0.5, 1.0\}$, β=0.5
`handle-pull-v3`	0%	12%	α=0.5, β=0.2
`pick-out-of-hole-v3`	0%	6%	multiple

Three of four tasks lift off zero under failure-only steering, with `dial-turn-v3` reaching 38% from a 0% baseline. The result is consistent with the transfer analysis in Section 4.4: failure geometry carries substantial task-relevant structure, and suppressing it can recover competent behavior without ever specifying where the policy should go. The effect is partial rather than complete, which matches the role of C^s in the full contrastive operator. We do not read this as evidence that failure suppression alone is sufficient in general, but as evidence that the negation term is doing nontrivial work within the contrastive recipe.

G Linear and Random Conceptor Ablation Analysis

Table 22 isolates two questions the main results in Table 1 leave open. The *linear* baseline replaces the multiplicative conceptor gate with the additive steering vector $h' = h + \alpha \cdot v$, where v is the unit mean-difference direction between success and failure activations. This is the standard contrastive activation-addition recipe (Rimsky et al., 2024; Postmus & Abreu, 2024) and asks whether the multiplicative, subspace-level form of the conceptor is load-bearing or whether a single-direction additive shift suffices. The *random* baseline replaces the conceptor's eigenvectors with a random orthonormal basis while preserving the eigenvalue spectrum, so the spectral filtering shape is matched but the identified directions are not. This asks whether gains come from any low-rank multiplicative gate or specifically from the contrastive success-minus-failure subspace that C_{steer} identifies.

Both ablations underperform the contrastive conceptor on every benchmark we tested. On GR00T N1.5 RoboCasa, the contrastive global gate improves mean success by $+0.16$ over the unsteered baseline, while linear steering reaches only $+0.03$ and random-direction steering moves the mean down by 0.02. On $\pi_{0.5}$ MetaWorld ML45 the gap widens: contrastive adds $+0.22$ over the 26 steered tasks, compared to $+0.10$ for linear and -0.03 for random. The per-task cells show the same ordering with occasional inversions: linear outperforms random on some RoboCasa tasks and trails on others, but neither baseline wins a single row against the contrastive gate. The random baseline tracking the unsteered baseline rules out the hypothesis that any low-rank multiplicative perturbation helps, and the linear baseline recovering roughly half of the contrastive gain shows that the mean-difference direction captures part of the discriminative signal but misses the higher-variance subspace directions that Boolean negation isolates. Taken together, both the multiplicative form and the specific contrastive subspace are necessary for the gains reported in Section 4.1.

H Per-Task Results with Selected Hyperparameters

Notes on Table 1. Table 1 compares the three COAST strategies of Sec. 3.2 (*global*, *per-step*, *positive-only*) against contrastive activation addition (+CAA; App. N.2), sparse-autoencoder steering (+SAE; TopK SAE feature selection following Khan et al. (2025) with the per-task best α over

Table 22: **Contrastive conceptors outperform both linear additive steering and random-direction controls on every cell where both baselines were run.** For each task we report the best success rate over a hyperparameter sweep (15 rollouts per cell). **Global**: contrastive conceptor $C_{\text{succ}} \wedge \neg C_{\text{fail}}$, shown as a reference for the main method (full results in Table 1). **Linear**: additive activation steering using the difference of mean success and failure activations. **Random**: random-direction conceptor with matched eigenvalue spectrum but random orthonormal eigenvectors, controlling for spectral filtering shape. **Bold** marks the best of the two ablated baselines per row.

(a) GR00T N1.5 on RoboCasa

Task	Base	Global	Linear	Random
Close Fridge	0.67	0.93	0.80	**0.85**
Coffee Mug	0.20	0.27	0.13	**0.23**
Open Drawer	0.53	0.80	**0.73**	0.67
Stand Mixer	0.60	0.80	**0.80**	0.60
PP Cabinet	0.73	0.80	0.53	**0.67**
PP Stove	0.73	0.87	**0.87**	0.60
Kettle	0.67	0.80	**0.47**	**0.47**
Mean	0.59	0.75	0.62	0.58
Δ	–	+0.16	+0.03	-0.02

(b) $\pi_{0.5}$ on MetaWorld ML45

Task	Base	Global	Linear	Random
pick-place-wall	0.20	0.87	0.40	**0.47**
assembly	0.27	1.00	0.00	**0.60**
stick-push	0.20	0.80	0.60	**0.73**
hammer	0.33	0.87	0.33	**0.80**
lever-pull	0.27	0.60	**0.80**	0.40
plate-slide-bs	0.60	0.93	0.60	**0.87**
disassemble	0.60	0.93	**0.73**	0.67
plate-slide-b	0.60	0.93	0.53	**0.87**
handle-pull-s	0.20	0.47	**0.47**	0.33
peg-insert-s	0.33	0.47	**0.53**	0.33
Steered (26)	0.58	0.80	0.68	0.55
Δ	–	+0.22	+0.10	−0.03
All (45)	0.73	0.86	0.79	0.72
Δ	–	+0.13	+0.06	−0.01

$\{0.25, 0.5, 1.0, 2.0\}$; App. N.3), and LoRA supervised fine-tuning on 30 on-policy rollouts per task (+SFT; App. N.1). CAA and SAE selected α post hoc on the test rollouts, which favors those baselines; matched-protocol re-runs are reported in App. B.2. LIBERO-10 task names abbreviate the manipulated objects (e.g., Stove+Moka = turn on the stove and put the moka pot on it); the full task names are listed in Table 23, and the KS/LR/ST codes used elsewhere in the appendix are mapped to these names in Table 12. All success rates are computed on 30 held-out test rollouts per task and condition, disjoint from the 15 fitting rollouts (App. K.5). Δ is the mean gain over the unsteered baseline; p_t is a paired t-test across tasks (the primary test) and p_z a pooled two-proportion z-test over episodes; Bonferroni- and Holm-corrected values are given in App. E. Bold marks the best method per row and underline the second best. †: the Cheese+Butter task has zero failures under the π_0-FAST checkpoint, so no contrastive conceptor can be fitted, and the means and Δ of the π_0-FAST LIBERO-10 columns are computed over the nine contrastive-eligible tasks. ‡: three or fewer successful episodes among the 15 fitting rollouts; the conceptor is nevertheless fit from hundreds of per-step activation vectors (App. F.3). The bar chart on the right of Table 1 reports real-robot success rates for $\pi_{0.5}$+COAST, $\pi_{0.5}$, π_0-FAST, and GR00T N1.5 on three tasks with 15 trials per task and condition; Wilson 95% confidence intervals and per-task hyperparameters are given in App. A.

Tables 23–28 report the per-task success rate on held-out test rollouts for the steering configuration that achieved the highest success rate on the fitting rollouts, under the full hyperparameter sweep described in Appendix I. For each entry, we also list the layer ℓ, aperture α, and steering strength β of the selected configuration. These tables back the aggregated numbers in Table 1 and expose two patterns that recur across model–benchmark pairs. First, the oracle-optimal layer concentrates in a narrow band: ℓ=11 (of 18) for the $\pi_{0.5}$ action expert and ℓ=10 (of 16) for the GR00T DiT, with occasional shifts to ℓ=5 on LIBERO tasks where the discriminative subspace forms earlier. This aligns with the quota peak used in Stage 1 of our geometric selection procedure (Sec. I.2). Second, small steering strengths dominate: $\beta \in \{0.1, 0.3\}$ attains the best fitting-set success rate in over 90% of cells, with β=0.5 preferred only on a handful of tasks where the baseline is already high and the conceptor acts as a light nudge. Together these tables justify the pruned grid used by geometric selection and show that the same small configuration neighborhood recovers near-oracle performance across tasks.

Table 23: Conceptor-based activation steering results of $\pi_0.5$ on LIBERO-10 with layer sweep in $\in \{5, 11\}$. Each condition is evaluated over 30 held-out test rollouts.

Task	Global	Per-Step	Pos.-Only
KS3: Turn On The Stove And Put The Moka Pot On It	**0.93** (L5, α=0.5, β=0.1)	**0.87** (L5, α=10.0, β=0.3)	**0.80** (L11, α=1.0, β=0.1)
KS4: Put The Black Bowl In The Bottom Drawer Of...	**0.73** (L5, α=1.0, β=0.1)	**0.80** (L5, α=0.1, β=0.1)	**0.87** (L5, α=1.0, β=0.1)
KS6: Put The Yellow And White Mug In The Microw...	**0.40** (L11, α=0.1, β=0.3)	**0.60** (L11, α=0.5, β=0.3)	**0.33** (L5, α=1.0, β=0.1)
KS8: Put Both Moka Pots On The Stove	**0.47** (L5, α=1.0, β=0.1)	**0.53** (L11, α=2.0, β=0.3)	0.13 (L5, α=1.0, β=0.1)
LR1: Put Both The Alphabet Soup And The Cream C...	**0.93** (L11, α=0.5, β=0.3)	**1.00** (L11, α=2.0, β=0.1)	**0.93** (L5, α=1.0, β=0.1)
LR2: Put Both The Alphabet Soup And The Tomato...	**0.80** (L11, α=0.5, β=0.3)	**0.80** (L5, α=2.0, β=0.1)	**0.60** (L5, α=1.0, β=0.3)
LR2: Put Both The Cream Cheese Box And The Butt...	**0.93** (L5, α=0.5, β=0.1)	**0.93** (L5, α=0.1, β=0.3)	**0.87** (L5, α=0.5, β=0.1)
LR5: Put The White Mug On The Left Plate And Pu...	**0.60** (L11, α=0.5, β=0.5)	**0.60** (L11, α=2.0, β=0.5)	**0.20** (L11, α=0.5, β=0.1)
LR6: Put The White Mug On The Plate And Put The...	**0.87** (L5, α=0.5, β=0.1)	**0.87** (L5, α=0.1, β=0.1)	**0.67** (L5, α=0.5, β=0.3)
ST1: Pick Up The Book And Place It In The Back...	**0.93** (L5, α=0.5, β=0.1)	**1.00** (L5, α=0.5, β=0.1)	**0.93** (L5, α=0.5, β=0.1)
Mean	**0.76**	**0.80**	**0.63**

Table 24: Conceptor-based activation steering results of π_0-FAST on LIBERO-10. LR2 (cream cheese & butter) has $n_{\text{fail}} = 0$ under this checkpoint, so contrastive conceptors (Global, Per-Step) cannot be constructed; means are computed over the 9 contrastive-eligible tasks.Each condition is evaluated over 30 held-out test rollouts.

Task	Global	Per-Step	Pos.-Only
KS3: Turn On The Stove And Put The Moka Pot On It	**1.00** (α=0.1, β=0.02)	**0.87** (α=0.1, β=0.3)	0.80 (α=0.1, β=0.05)
KS4: Put The Black Bowl In The Bottom Drawer Of...	**0.93** (α=2.0, β=0.1)	**0.93** (α=0.1, β=0.2)	**0.80** (α=0.5, β=0.1)
KS6: Put The Yellow And White Mug In The Microw...	**0.73** (α=10.0, β=0.3)	**0.80** (α=0.5, β=0.1)	**0.73** (α=2.0, β=0.1)
KS8: Put Both Moka Pots On The Stove	**0.53** (α=5.0, β=0.15)	**0.53** (α=3.0, β=0.2)	**0.40** (α=3.0, β=0.01)
LR1: Put Both The Alphabet Soup And The Cream C...	**0.87** (α=10.0, β=0.2)	**0.87** (α=0.1, β=0.2)	**1.00** (α=0.1, β=0.1)
LR2: Put Both The Alphabet Soup And The Tomato...	**0.87** (α=2.0, β=0.2)	**0.93** (α=0.1, β=0.2)	**0.73** (α=0.2, β=0.05)
LR2: Put Both The Cream Cheese Box And The Butt...	– (no C_{fail})	– (no C_{fail})	1.00 (α=0.5, β=0.1)
LR5: Put The White Mug On The Left Plate And Pu...	**0.87** (α=1.0, β=0.2)	**0.87** (α=0.1, β=0.3)	**0.80** (α=0.1, β=0.05)
LR6: Put The White Mug On The Plate And Put The...	**0.93** (α=1.0, β=0.1)	**0.93** (α=0.3, β=0.05)	**0.87** (α=0.1, β=0.1)
ST1: Pick Up The Book And Place It In The Back...	**0.87** (α=10.0, β=0.3)	**0.87** (α=0.07, β=0.1)	**0.73** (α=0.07, β=0.12)
Mean	**0.84**	**0.84**	**0.79**

Table 25: Conceptor-based activation steering results of $\pi_{0.5}$ on RoboCasa with layer sweep $\in \{5, 11\}$. Each condition is evaluated over 30 held-out test rollouts.

Task	Global	Per-Step	Pos.-Only
Close Fridge	**0.47** (L11, α=0.5, β=0.3)	**0.40** (t=0, L11, α=0.1, β=0.5)	**0.40** (L5, α=2.0, β=0.1)
Coffee Setup Mug	**0.33** (L11, α=0.1, β=0.1)	**0.33** (t=0, L5, α=0.5, β=0.1)	**0.53** (L5, α=0.5, β=0.5)
Open Drawer	**0.67** (L11, α=0.5, β=0.1)	0.53 (t=0, L5, α=0.5, β=0.1)	**0.67** (L5, α=1.0, β=0.3)
Open Stand Mixer Head	**0.73** (L11, α=0.5, β=0.1)	**0.80** (t=9, L11, α=1.0, β=0.1)	0.53 (L5, α=0.1, β=0.3)
Pick Place Counter To Cabinet	**0.73** (L5, α=0.1, β=0.1)	**0.73** (t=0, L5, α=0.5, β=0.1)	**0.67** (L5, α=10.0, β=0.3)
Pick Place Counter To Stove	**0.53** (L11, α=0.1, β=0.1)	**0.67** (t=0, L11, α=1.0, β=0.1)	**0.60** (L5, α=0.5, β=0.1)
Turn On Electric Kettle	**0.40** (L11, α=0.5, β=0.1)	**0.40** (t=0, L11, α=2.0, β=0.1)	**0.53** (L5, α=0.5, β=0.3)
Mean	**0.55**	**0.55**	**0.56**

I Hyperparameter Sweep Ranges and Efficient Selection

I.1 Full grid specification

Strategies.

- **Global**: A single contrastive conceptor C_{contr} applied identically at all denoising steps.
- **Per-step**: A separate contrastive conceptor per denoising step, fitted from activations at that specific step.
- **Positive-only**: Uses C_s directly (no contrastive NOT operation), i.e. $M = (1-\beta)I + \beta\, C_s$.
- **Linear (ActAdd)**: $h' = h + \alpha \cdot v$, where $v = \text{unit}(\bar{h}_s - \bar{h}_f)$ is the unit mean-difference direction. Here α controls the perturbation magnitude.
- **Random**: Random PSD matrix baseline to control for any generic projection effect.

I.2 Efficiently selecting hyperparameters with minimal rollouts

COAST introduces three hyperparameters: the injection layer ℓ, the aperture α, and the steering strength β. A naive grid search across our ranges costs up to 152 configurations per task (Table 30),

Table 26: **Conceptor steering results of $\pi_0.5$ on MetaWorld ML45 (all 26 steered tasks).** For each task and steering strategy, we report the best success rate over a hyperparameter sweep of $\alpha \in \{0.1, 0.5, 1.0\}$ and $\beta \in \{0.1, 0.3, 0.5\}$ at layer 11. **Global**: contrastive conceptor ($C_{\text{succ}} \wedge \neg C_{\text{fail}}$) applied uniformly across denoise steps. **Per-Step**: contrastive conceptor applied independently at each denoise step. **Pos.-Only**: success conceptor without failure subtraction. Best hyperparameters shown in parentheses. Each condition is evaluated over 30 held-out test rollouts.

Task	Baseline	Global	Per-Step	Pos.-Only
assembly	0.27	**1.00** (α=0.5, β=0.1)	**0.80** (α=0.5, β=0.1)	0.27 (α=0.1, β=0.5)
basketball	0.40	**0.47** (α=0.1, β=0.1)	**0.60** (α=0.1, β=0.1)	**0.60** (α=0.5, β=0.1)
coffee-pull	0.93	0.93 (α=0.1, β=0.1)	0.93 (α=1.0, β=0.1)	**1.00** (α=0.1, β=0.3)
coffee-push	0.80	**1.00** (α=1.0, β=0.1)	**1.00** (α=0.5, β=0.1)	**1.00** (α=0.1, β=0.1)
disassemble	0.60	**0.93** (α=0.1, β=0.1)	**0.93** (α=0.1, β=0.1)	**0.87** (α=0.5, β=0.5)
door-open	0.93	**1.00** (α=0.1, β=0.1)	**1.00** (α=0.1, β=0.1)	**1.00** (α=0.1, β=0.1)
faucet-close	0.80	**1.00** (α=1.0, β=0.1)	**0.93** (α=0.1, β=0.1)	**0.93** (α=0.1, β=0.1)
hammer	0.33	**0.87** (α=0.5, β=0.1)	**0.87** (α=0.1, β=0.1)	**0.40** (α=0.1, β=0.1)
handle-pull-side	0.20	**0.47** (α=0.1, β=0.1)	**0.33** (α=0.5, β=0.1)	**0.53** (α=0.5, β=0.1)
handle-pull	0.47	0.47 (α=0.1, β=0.1)	**0.60** (α=0.1, β=0.1)	**0.53** (α=0.1, β=0.1)
lever-pull	0.27	**0.60** (α=0.1, β=0.1)	**0.67** (α=0.1, β=0.3)	**0.73** (α=0.5, β=0.3)
peg-insert-side	0.33	**0.47** (α=0.1, β=0.3)	**0.67** (α=0.1, β=0.3)	**0.53** (α=0.1, β=0.3)
pick-out-of-hole	0.20	**0.27** (α=0.1, β=0.3)	**0.47** (α=0.1, β=0.1)	**0.47** (α=1.0, β=0.1)
pick-place	0.87	**1.00** (α=0.5, β=0.1)	**1.00** (α=0.1, β=0.1)	0.80 (α=0.1, β=0.3)
pick-place-wall	0.20	**0.87** (α=0.5, β=0.1)	**1.00** (α=0.1, β=0.1)	**0.47** (α=1.0, β=0.3)
plate-slide-back-side	0.60	**0.93** (α=1.0, β=0.1)	**1.00** (α=0.1, β=0.3)	0.60 (α=1.0, β=0.1)
plate-slide-back	0.60	**0.93** (α=0.1, β=0.1)	**0.93** (α=1.0, β=0.1)	**0.73** (α=0.1, β=0.1)
push-back	0.67	**0.93** (α=0.1, β=0.1)	**0.87** (α=1.0, β=0.3)	**0.87** (α=0.1, β=0.3)
push	0.93	**1.00** (α=0.1, β=0.1)	**1.00** (α=1.0, β=0.1)	0.93 (α=1.0, β=0.1)
reach	0.93	0.93 (α=0.1, β=0.1)	0.93 (α=0.1, β=0.1)	**1.00** (α=0.1, β=0.1)
shelf-place	0.73	**0.93** (α=0.5, β=0.1)	**0.93** (α=0.5, β=0.1)	**0.93** (α=0.1, β=0.3)
soccer	0.27	**0.40** (α=0.1, β=0.3)	**0.40** (α=1.0, β=0.5)	**0.40** (α=0.1, β=0.5)
stick-pull	0.73	**0.87** (α=1.0, β=0.1)	**0.87** (α=0.1, β=0.1)	**0.87** (α=0.1, β=0.1)
stick-push	0.20	**0.80** (α=0.1, β=0.1)	**0.73** (α=1.0, β=0.1)	**0.40** (α=0.1, β=0.3)
sweep-into	0.73	**0.87** (α=0.5, β=0.1)	**0.87** (α=0.1, β=0.1)	**0.93** (α=1.0, β=0.1)
sweep	0.93	**1.00** (α=1.0, β=0.1)	**1.00** (α=0.5, β=0.1)	**1.00** (α=0.1, β=0.1)
Steered tasks (26)	0.57	0.81	0.82	0.72
Δ vs. Baseline	—	+0.23	+0.25	+0.15

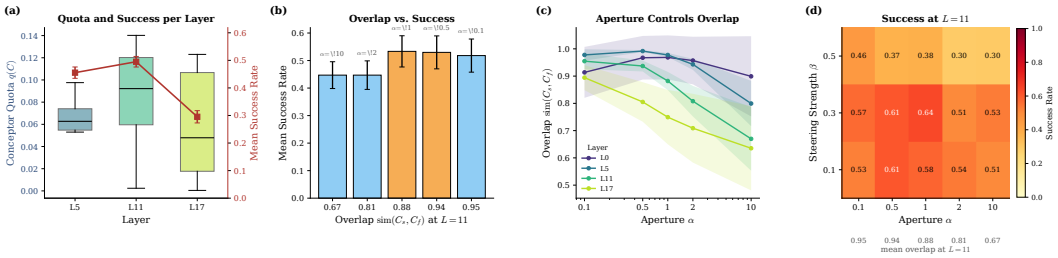


Figure 9: **Three-stage hyperparameter selection, validated on LIBERO-10 with $\pi_{0.5}$. (A)** Quota $q(\mathbf{C})$ across layers (boxes) and mean steered success (red) both peak at L=11, so quota alone identifies the correct layer. **(B)** Mean success versus overlap at L=11. Success rises as overlap decreases, then saturates. **(C)** Overlap decreases monotonically with aperture α, so selecting the overlap band determines a narrow α range with no rollouts. **(D)** Success heatmap over $\alpha \times \beta$ at L=11, confirming that small β values perform comparably, leaving 2–3 configurations to evaluate per task.

which scales poorly across the dozens of tasks in our benchmarks. The same geometric quantities that explain the mechanism in Section 4.3 also select these hyperparameters in closed form, narrowing the search to a small number of configurations at negligible additional GPU cost.

Table 27: **Conceptor steering results for π_0-FAST on MetaWorld ML45 (all 27 steered tasks).** For each task and steering strategy, we report the best success rate over an extended hyperparameter sweep at layer 17. The grids are $\alpha_{\text{global}} \in \{0.1, 0.5, 1, 2, 10\}$, $\alpha_{\text{per-step}} \in \{0.1, 0.5, 1\}$, $\alpha_{\text{pos.-only}} \in \{0.1, 0.5, 1, 2\}$, and $\beta \in \{0.1, 0.2, 0.3, 0.5\}$. **Global**: contrastive conceptor ($C_{\text{succ}} \wedge \neg C_{\text{fail}}$) applied uniformly across denoise steps. **Per-Step**: contrastive conceptor applied independently at each denoise step. **Pos.-Only**: success conceptor without failure subtraction. Best hyperparameters shown in parentheses. Each condition is evaluated over 16 rollouts. Bold marks values strictly above the unsteered baseline.

Task	Baseline	Global	Per-Step	Pos.-Only
basketball	0.62	0.62 (α=10, β=0.1)	**0.69** (α=0.1, β=0.1)	**0.75** (α=0.1, β=0.2)
coffee-pull	0.62	**0.69** (α=1.0, β=0.3)	0.62 (α=0.1, β=0.2)	**0.69** (α=0.5, β=0.2)
coffee-push	0.88	**0.94** (α=1.0, β=0.3)	**0.94** (α=0.1, β=0.2)	**0.94** (α=2.0, β=0.1)
disassemble	0.56	**0.69** (α=1.0, β=0.3)	0.56 (α=0.1, β=0.1)	**0.75** (α=0.5, β=0.5)
door-close	0.81	**0.94** (α=2.0, β=0.3)	**0.94** (α=0.1, β=0.1)	**1.00** (α=0.1, β=0.3)
faucet-close	0.44	**0.69** (α=0.5, β=0.1)	**0.69** (α=0.1, β=0.5)	**0.81** (α=0.1, β=0.1)
faucet-open	0.81	**0.94** (α=2.0, β=0.5)	**0.88** (α=0.5, β=0.2)	**0.88** (α=2.0, β=0.3)
hammer	0.69	0.69 (α=0.5, β=0.2)	0.62 (α=0.1, β=0.2)	0.62 (α=2.0, β=0.3)
handle-press-side	0.94	**1.00** (α=1.0, β=0.1)	**1.00** (α=0.1, β=0.2)	**1.00** (α=0.5, β=0.2)
handle-pull-side	0.12	**0.25** (α=0.5, β=0.5)	**0.25** (α=0.5, β=0.2)	**0.19** (α=0.5, β=0.1)
lever-pull	0.19	**0.56** (α=0.5, β=0.5)	**0.44** (α=0.1, β=0.5)	**0.56** (α=0.5, β=0.2)
peg-insert-side	0.19	**0.31** (α=0.5, β=0.1)	**0.31** (α=1.0, β=0.3)	0.19 (α=1.0, β=0.1)
peg-unplug-side	0.69	**1.00** (α=2.0, β=0.5)	**0.94** (α=0.1, β=0.2)	**0.81** (α=0.5, β=0.2)
pick-place	0.69	**0.88** (α=0.5, β=0.5)	**0.88** (α=1.0, β=0.3)	**0.81** (α=0.5, β=0.1)
pick-place-wall	0.69	**0.81** (α=1.0, β=0.1)	**0.75** (α=0.1, β=0.3)	**0.75** (α=0.1, β=0.1)
plate-slide-back	0.88	0.88 (α=0.5, β=0.3)	0.88 (α=1.0, β=0.3)	**0.94** (α=0.5, β=0.1)
push-back	0.62	**0.69** (α=2.0, β=0.2)	0.62 (α=0.1, β=0.3)	**0.69** (α=1.0, β=0.3)
push	0.75	**0.94** (α=2.0, β=0.3)	**0.88** (α=0.1, β=0.3)	0.75 (α=0.5, β=0.3)
push-wall	0.06	**0.19** (α=10, β=0.2)	**0.25** (α=1.0, β=0.3)	**0.25** (α=0.5, β=0.1)
reach	0.81	0.81 (α=1.0, β=0.1)	0.81 (α=0.1, β=0.1)	0.81 (α=0.5, β=0.1)
reach-wall	0.56	**0.69** (α=2.0, β=0.2)	**0.69** (α=0.5, β=0.3)	**0.75** (α=0.1, β=0.5)
shelf-place	0.19	**0.56** (α=2.0, β=0.3)	**0.56** (α=1.0, β=0.3)	**0.44** (α=1.0, β=0.1)
soccer	0.56	**0.62** (α=1.0, β=0.1)	0.56 (α=0.5, β=0.1)	0.56 (α=2.0, β=0.1)
stick-pull	0.25	**0.44** (α=2.0, β=0.1)	**0.38** (α=0.5, β=0.5)	**0.44** (α=2.0, β=0.3)
stick-push	0.81	**0.88** (α=0.1, β=0.2)	**0.88** (α=0.5, β=0.3)	**0.88** (α=0.5, β=0.1)
sweep-into	0.44	**0.56** (α=10, β=0.1)	**0.56** (α=0.5, β=0.1)	**0.50** (α=1.0, β=0.3)
sweep	0.44	**0.50** (α=1.0, β=0.3)	**0.50** (α=1.0, β=0.3)	**0.56** (α=1.0, β=0.2)
Steered tasks (27)	0.57	0.70	0.67	0.68
Δ vs. Baseline	—	+0.13	+0.10	+0.11

Table 28: Conceptor-based activation steering results on RoboCasa with GR00T N1.5, showing the selected layer, aperture α, and mixing weight β for the best configuration of each strategy. **Bold** indicates the best performance. Each condition is evaluated over 30 held-out test rollouts.

Task	Global	Per-Step	Pos.-Only
Close Fridge	0.93 (L10, α=1.0, β=0.1)	**1.00** (L10, β=0.2)	0.87 (L10, α=0.1, β=0.1)
Coffee Setup Mug	0.27 (L10, α=0.5, β=0.1)	**0.33** (L10, β=0.2)	**0.33** (L10, α=1.0, β=0.3)
Open Drawer	**0.80** (L10, α=0.5, β=0.1)	0.53 (L10, β=0.1)	0.67 (L10, α=0.1, β=0.3)
Open Stand Mixer Head	0.80 (L10, α=0.1, β=0.1)	0.80 (L10, β=0.1)	**0.93** (L10, α=0.1, β=0.3)
Pick Place Counter To Cabinet	**0.80** (L10, α=0.3, β=0.5)	**0.80** (L10, β=0.5)	0.73 (L10, α=0.1, β=0.3)
Pick Place Counter To Stove	**0.87** (L10, α=0.1, β=0.1)	0.75 (L10, β=0.1)	**0.87** (L10, α=1.0, β=0.3)
Turn On Electric Kettle	**0.80** (L10, α=0.8, β=0.08)	0.73 (L10, β=0.08)	0.73 (L10, α=0.8, β=0.05)
Mean	**0.75**	0.71	0.73

Stage 1: Layer selection via quota. Both conceptor quota $q(\mathbf{C}) = \frac{1}{d}\text{tr}(\mathbf{C})$ and steered success are unimodal in depth, peaking at layer 11 (Figure 9A). Early layers encode features too generic

Table 29: Hyperparameter sweep grid per model–benchmark combination.

Parameter	pi0.5 MetaWorld	pi0.5 LIBERO	pi0.5 RoboCasa	GR00T RoboCasa
Layers ℓ	$\{0, 5, 11, 17\}$	$\{0, 5, 11, 17\}$	$\{0, 5, 11, 17\}$	$\{0, 5, 10, 15\}$
Aperture α	$\{0.1, 0.5, 1, 2, 10\}$	$\{0.1, 0.5, 1, 2, 10\}$	$\{0.1, 0.5, 1, 2, 10\}$	$\{0.1, 0.3, 0.5, 0.8, 1, 1.5, 2, 3, 5, 10\}$
Strength β	$\{0.1, 0.3, 0.5\}$	$\{0.1, 0.3, 0.5\}$	$\{0.1, 0.3, 0.5\}$	$\{0.1, 0.3, 0.5\}$
Strategy	global, per-step, positive-only, linear	global, per-step, positive-only, linear	global, per-step	global, per-step, positive-only, linear
Full grid / task	$4 \times 5 \times 3 \times 4 = 240$	$4 \times 5 \times 3 \times 4 = 240$	$4 \times 5 \times 3 \times 2 = 120$	$4 \times 5 \times 3 \times 4 = 240$

Table 30: Oracle gap analysis. Geometric selection achieves 86–93% of oracle performance while evaluating only a small fraction of the full configuration grid. Selected parameters: GR00T N1.5 RoboCasa uses ℓ=10, α=0.1; $\pi_{0.5}$ RoboCasa uses ℓ=11, α=0.5; $\pi_{0.5}$ LIBERO uses ℓ=11, $\alpha \in \{0.5, 1.0\}$. All benchmarks use $\beta \in \{0.1, 0.3\}$.

Benchmark	Tasks	Full Grid	Selected Grid	% Grid Evaluated	% of Oracle SR
GR00T N1.5 RoboCasa	7	240	7	2.9%	92.8%
$\pi_{0.5}$ RoboCasa	7	120	10	8.3%	93.4%
$\pi_{0.5}$ LIBERO	10	240	18	7.5%	92.1%

to form a discriminative subspace, while late layers encode features too state-specific for a global conceptor to pool cleanly. Because quota aligns with success across the full sweep, we select the layer from quota alone and fix $\ell^\star = 11$ at zero GPU cost.

For each candidate layer ℓ, compute the mean contrastive-conceptor *quota* across tasks:

$$q(\ell) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \frac{1}{d} \operatorname{tr}(C_{\text{contr}}^{(t,\ell,\alpha_0)}), \tag{6}$$

using the largest aperture α_0=10 (which gives the sharpest conceptor). Select the layer with the highest mean quota: $\ell^* = \arg\max_\ell q(\ell)$.

Stage 2: Aperture selection via overlap. With the layer fixed, aperture selection follows from the overlap finding of Section 4.3. Figure 9B shows that steered success rises as overlap decreases from near-identity and then saturates. Below the saturation point the contrastive signal destabilizes the policy, and above it the two conceptors are nearly identical so the signal vanishes. Because overlap decreases monotonically with aperture (Figure 9C), the target overlap band determines a narrow α range, and computing overlap at each candidate α requires a single $d \times d$ matrix product with no rollouts.

At the selected layer ℓ^*, compute the mean success–failure overlap for each α:

$$\text{overlap}(\alpha) = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \frac{\operatorname{tr}(C_s^{(t)} \cdot C_f^{(t)})}{\sqrt{\operatorname{tr}((C_s^{(t)})^2)\ \operatorname{tr}((C_f^{(t)})^2)}}. \tag{7}$$

Retain alphas whose mean overlap falls in a "sweet spot" band $[0.85, 0.95]$. If no alpha falls in the band, take the closest one.

Stage 3: Steering strength via narrow sweep. Only β remains. Figure 9D confirms that large β degrades performance uniformly while small values perform comparably, leaving at most two or three configurations to evaluate per task. This is the only stage that requires rollouts.

Universally drop β=0.5 (which was found to be harmful across all benchmarks in preliminary experiments) and keep $\beta \in \{0.1, 0.3\}$.

Validation: oracle gap analysis. To quantify the practical value of geometric selection, we compare the configuration chosen by the three-stage procedure against the best configuration found by exhaustive sweep across the full hyperparameter grid (Table 30). Across all three benchmark–model

pairs, geometric selection recovers 93% of oracle success rate while evaluating only 3–8% of the grid. These results confirm that the geometric quantities from Section 4.3—quota for layer selection, overlap for aperture selection—are not merely descriptive but operationally sufficient for near-optimal configuration at an order-of-magnitude reduction in evaluation cost.

To avoid evaluating the full grid (which requires >100 rollouts per task), we use a three-step geometric selection procedure based solely on the pre-computed conceptor matrices:

Grid reduction. This procedure reduces the sweep from (e.g.) 240 conditions/task to ∼4–12 conditions/task (a 20×–60× reduction), making per-task evaluation tractable.

Table 31: Selected parameters per model–benchmark (geometric procedure).

Model–Benchmark	**Layer ℓ^***	**Alphas**	**Betas**
pi0.5 LIBERO-10	11	$\{0.5, 1.0\}$	$\{0.1, 0.3\}$
pi0.5 RoboCasa	11	$\{0.5\}$	$\{0.1, 0.3\}$
GR00T N1.5 RoboCasa	10	$\{0.1\}$	$\{0.1, 0.3\}$

I.3 Per-task selected hyperparameters

Table 32 shows the per-task oracle-selected hyperparameters (best rollout success rate) for LIBERO-10.

Table 32: Per-task oracle-selected hyperparameters for pi0.5 LIBERO-10.

Task (abbreviated)	**Layer**	α	β
KITCHEN_SCENE3 (stove + moka pot)	5	0.5	0.1
KITCHEN_SCENE4 (bowl in drawer)	5	1.0	0.1
KITCHEN_SCENE6 (mug in microwave)	11	0.1	0.3
KITCHEN_SCENE8 (two moka pots)	5	1.0	0.1
LIVING_ROOM1 (soup + cheese in basket)	11	0.5	0.3
LIVING_ROOM2 (soup + tomato sauce in basket)	11	0.5	0.3
LIVING_ROOM2 (cheese + butter in basket)	5	0.5	0.1
LIVING_ROOM5 (two mugs on plates)	11	0.5	0.5
LIVING_ROOM6 (mug + pudding)	5	0.5	0.1
STUDY_SCENE1 (book in caddy)	5	0.5	0.1

Geometric vs. oracle gap. The geometric procedure selects layer 11 with $\alpha \in \{0.5, 1.0\}$ and $\beta \in \{0.1, 0.3\}$. This matches the oracle layer for 5/10 tasks (those where layer 11 is also oracle-optimal). For the remaining 5 tasks where the oracle prefers layer 5, the geometric procedure's choice of layer 11 may yield slightly lower success rates, but the gap is typically <5 percentage points because the layer-5 and layer-11 conceptors have similar quotas.

I.4 Diagnostic quantities

Table 33: Layer quotas (mean across tasks, α=10) for pi0.5 LIBERO.

	Layer 0	Layer 5	Layer 11	Layer 17
Mean quota	0.016	0.059	**0.092**	0.082

J Activation Extraction Details

We extract intermediate activations from three model families: **pi0.5** (a PaLI-Gemma 2 backbone with a Gemma 2 action-expert suffix) and **GR00T N1.5** (an SigLIP/T5 vision-language backbone with a DiT action head). Both are flow-matching diffusion policies that produce action chunks via an Euler denoising loop. Below we detail the extraction procedure for each.

Table 34: Mean success–failure overlap at layer 11 for pi0.5 LIBERO.

α	0.1	0.5	1.0	2.0	10.0
Overlap	0.955	**0.937**	**0.882**	0.808	0.670

Algorithm 1 Steering on a Generic DiT Action Head

Require: Conditioning tokens c, robot state s, denoising/flow steps K, transformer layers L, hook layer ℓ_{hook}, conceptor C_{steer}, strength β

$M \leftarrow (1-\beta)I + \beta C_{\text{steer}}$
$x \sim \mathcal{N}(0, I), \quad t \leftarrow 1, \quad \Delta t \leftarrow -1/K$
for $k = 1, \ldots, K$ **do**
  $h^0 \leftarrow \phi_{\text{in}}(x, s, t)$
  for $\ell = 1, \ldots, L$ **do**
    $u^\ell \leftarrow \text{Attn}^\ell(h^{\ell-1}, c)$
    $h^\ell \leftarrow \text{MLP}^\ell(u^\ell)$ ▷ layer output
    if $\ell = \ell_{\text{hook}}$ **then**
      $h^\ell \leftarrow h^\ell M^\top$
    end if
  end for
  $v_t \leftarrow \phi_{\text{out}}(h^L)$
  $x \leftarrow \text{Step}(x, v_t, t, \Delta t)$
  $t \leftarrow t + \Delta t$
end for
return $\hat{a} \leftarrow x$

J.1 $\pi_{0.5}$ (METAWORLD, LIBERO, ROBOCASA)

Architecture context. pi0.5 is a two-tower model. A frozen PaLI-Gemma 2 backbone encodes images and language into a prefix KV cache. A trainable *Gemma 2 action expert* (18 decoder layers, hidden dimension d=1024) operates on a suffix sequence whose tokens represent the current proprioceptive state and a noised action chunk. The expert layers use Adaptive RMS normalization (AdaRMS), where a pooled conditioning vector gates each layer's input-layernorm (attention) and post-attention-layernorm (MLP) via learned scale factors.

Denoising loop. Inference runs 10 Euler denoising steps. At each step $k \in \{0, \ldots, 9\}$, the noise schedule is a linear interpolation from t=1 (pure noise) to t=0 (clean actions), with uniform step size $\Delta t = -1/10$:

$$x_{k+1} = x_k + \Delta t \cdot v_\theta(x_k, t_k), \qquad t_k = 1 + k\,\Delta t. \tag{8}$$

All 10 steps are executed; no early stopping is used.

Hook registration (V1, used for conceptor fitting). The model method `sample_actions_with_intermediates` registers PyTorch forward hooks on the action-expert decoder layers `paligemma_with_expert.gemma_expert.model.layers[i]` for $i \in \{0, 5, 11, 17\}$ (four equally-spaced layers spanning the full 18-layer stack). Two hooks per layer:

1. **Residual stream** (output hook on `expert_layers[i]`): captures the full layer output $h_i^{(\text{out})} \in \mathbb{R}^{B \times S \times 1024}$. If the output is a tuple, only the first element (hidden state) is retained.
2. **MLP hidden state** (output hook on `expert_layers[i].mlp.down_proj` that captures the *input* to that module): captures the post-GELU expanded activation $m_i \in \mathbb{R}^{B \times S \times 4096}$, i.e. the input to the down-projection linear layer.

All hooks capture at *every* denoising step and move tensors to CPU immediately via `.detach().cpu()`.

Hook registration (V2, selective). A second extraction variant, `sample_actions_with_intermediates_v2`, adds two additional activation types and supports selective step collection:

1. **Attention weights**: A forward pre-hook injects `output_attentions=True` into the layer's kwargs, and a forward output hook captures the attention weight matrix $A_i \in \mathbb{R}^{B\times H\times S\times S}$ (where H is the number of heads).
2. **AdaRMS gates**: Forward hooks on `expert_layers[i].input_layernorm` (attention gate) and `expert_layers[i].post_attention_layernorm` (MLP gate) capture the per-token gating vector $g_i \in \mathbb{R}^{B\times 1\times 1024}$. These are collected for *all* 18 expert layers. Non-adaptive layers (if any) produce zero tensors.

V2 only records intermediates at user-specified denoising steps (default $\{0, 4, 9\}$) to reduce storage; the full 10-step loop still runs.

AdaRMS conditioning vector. The AdaRMS conditioning vector $c \in \mathbb{R}^{B\times 1024}$ is computed from the proprioceptive state and the current timestep embedding inside `embed_suffix`. In V1 it is captured at every denoising step; in V2 only the step-0 value is saved (it is deterministic given fixed observations and noise).

Mean-pooling. Before computing conceptors, activations are mean-pooled over the token dimension. For **pi0.5 on MetaWorld / LIBERO**, the suffix sequence contains $S{=}10$ action-horizon tokens (no padding); the mean is over all 10 positions. For **pi0.5 on RoboCasa**, S depends on the config's action horizon. No padding mask is applied because all suffix tokens are valid action tokens. After pooling, each inference step yields one vector $\bar{h} \in \mathbb{R}^{1024}$ per layer per denoising step.

GR00T N1.5 token pooling. For GR00T N1.5 (see below), the self-attention sequence has $S{=}49$ tokens (1 state token + 32 future tokens + 16 action tokens). Mean-pooling is over all 49 positions.

Precision. pi0.5 runs inference in the model's native `bfloat16` precision. Residual-stream and MLP-hidden activations are cast to `float32` before being written as NumPy arrays. The denoising trajectory (x_t, v_t) is always `float32`. AdaRMS conditioning and attention weights are also stored in `float32`.

Conceptor covariance matrices $R = \tilde{X}^\top \tilde{X}/N$ are computed in `float64` (NumPy's default) and stored as `float32` in the compressed `.npz` file. No eigenvalue truncation is applied; the full $d \times d$ matrix is retained.

J.2 GR00T N1.5 (RoboCasa)

Architecture context. GR00T N1.5 uses a SigLIP + T5 vision-language backbone that produces a variable-length context sequence, followed by a 16-layer DiT (Diffusion Transformer) action head with hidden dimension $d{=}1536$ and feed-forward inner dimension $F{=}6144$. Unlike pi0.5's AdaRMS, GR00T conditions the DiT on the backbone via *cross-attention* from a full VL sequence computed once per inference call.

Denoising loop. GR00T uses 4 Euler denoising steps with uniform step size $\Delta t = 1/4$. Timesteps are discretized into buckets for the DiT's learned timestep embedding.

Hook registration. The function `_get_action_with_intermediates` in `groot_adapter.py` registers two hooks per DiT layer (`head.model.transformer_blocks[i]`, $i \in \{0, \ldots, 15\}$):

1. **Residual stream** (output hook on each `transformer_blocks[i]`): captures $h_i^{(\text{out})} \in \mathbb{R}^{B\times 49\times 1536}$.
2. **MLP hidden state** (output hook on `transformer_blocks[i].ff.net[2]` that captures the *input*): captures the post-GELU expanded activation $m_i \in \mathbb{R}^{B\times 49\times 6144}$.

Both hook types fire at all 4 denoising steps.

Backbone features. The VL backbone output vl_embs $\in \mathbb{R}^{B \times S_{\text{vl}} \times 1536}$ (where S_{vl} is the variable VL sequence length) is saved once per inference call as `backbone_cond.npz`. This is GR00T's analog of pi0.5's AdaRMS conditioning, but differs architecturally: pi0.5 produces a *pooled* 1024-dim vector per denoising step, while GR00T produces a variable-length sequence used via cross-attention.

Precision. GR00T runs under `torch.autocast(dtype=bfloat16)` on CUDA. DiT hidden states and MLP hidden states are cast to `float16` before writing. Denoising trajectories and backbone features follow the same `float32` / `float16` convention as pi0.5.

On-disk storage schema (both models). Each inference step's activations are saved under a directory tree:

```
<output_dir>/<checkpoint_step>/<task_name>/
  episode_NNN_env_NNN/
    metadata.json          # episode_success, total_reward, ...
    rewards.npz            # per_step_reward, cumulative_reward, success
    step_NNNN/
      metadata.json        # step, inference_step, cumulative_reward, ...
      denoising.npz        # all_x_t (D,H,A), all_v_t (D,H,A) fp32
      suffix_residual.npz  # (D,L,S,1024) fp32  [pi0.5]
      suffix_mlp_hidden.npz# (D,L,S,4096) fp32  [pi0.5]
      adarms_cond.npz      # (D,1024) fp32       [pi0.5 V1]
      dit_hidden_states.npz# (D,L,S,1536) fp16  [GR00T]
      dit_mlp_hidden.npz   # (D,L,S,6144) fp16  [GR00T]
      backbone_cond.npz    # (S_vl,1536) fp16   [GR00T]
```

where D=denoising steps, L=layers, S=sequence length, H=action horizon, A=action dimension.

K Rollout Collection Protocol

Activation data for conceptor fitting is collected by executing the policy in each environment and saving intermediate activations at every inference step. For every (model, checkpoint, benchmark) combination, we collect two strictly disjoint sets of rollouts: 15 *fitting* rollouts per task, used to construct conceptors and select hyperparameters, and 30 *test* rollouts per task, used exclusively to evaluate the selected steering configuration. The two sets use different environment seeds, object placements, and initial conditions, and share no overlap in episode identifiers. All success rates reported in the main paper are computed on the test rollouts only. Below we describe the per-benchmark details.

K.1 Common infrastructure

All environments use a server–client WebSocket architecture. The policy server loads the model and serves actions; the environment client sends observations and receives actions. For activation collection, the server is started in *collection mode*, which wraps the policy in a `CollectingPolicy` that:

1. Rejects plain inference requests (ensures the server cannot accidentally serve evaluation traffic).
2. Requires each request to carry a `__collect__` payload (per-step metadata: task name, episode id, env id, step counter, cumulative reward) or a `__finalize_episode__` payload (episode-level metadata: total reward, success flag, per-step rewards).
3. Calls `infer_with_intermediates` on the underlying model, which runs the denoising loop with forward hooks registered.
4. Writes per-step `.npz` files and per-episode `metadata.json` / `rewards.npz`.

K.2 METAWORLD (PI0.5 AND π_0-FAST)

- **Tasks**: 45 ML45 training tasks (or individual tasks).
- **Fitting rollouts**: 15 parallel environments (`-num_envs 15`), each run for up to 300 environment steps (replan every 32 steps $\Rightarrow \sim$10 inference steps per episode). These rollouts provide the success and failure activations from which conceptors are constructed and hyperparameters are selected.
- **Test rollouts**: 30 rollouts per task per steering condition, collected with fresh environment seeds disjoint from the fitting set. All success rates in Table 1 and Tables 26–27 are computed on these rollouts only.
- **Initial conditions**: MetaWorld's built-in random initialization per environment seed. Seeds are set by the `env_id` index.
- **Success labeling**: MetaWorld's native per-step `info["success"]` flag. An episode is labeled successful if any step achieves `success=True`.
- **Checkpoint used**: `/openpi-metaworld-5000` (gradient step 5,000 of 30,000 total), an early checkpoint with mixed success/failure outcomes, necessary for contrastive conceptor fitting.
- **Typical success/failure split**: Varies by task; the early checkpoint yields approximately 30–60% success rate on most ML45 training tasks, ensuring both classes are represented.

K.3 LIBERO-10 (PI0.5 AND π_0-FAST)

- **Tasks**: 10 LIBERO-10 tasks (full names in Table 23).
- **Fitting rollouts**: 15 episodes per task (`-num_episodes 15`). These provide the activations for conceptor construction and hyperparameter selection.
- **Test rollouts**: 30 episodes per task per steering condition, collected with fresh seeds disjoint from the fitting set.
- **Initial conditions**: LIBERO's built-in per-task initial state distribution. The environment is reset with sequential episode seeds.
- **Success labeling**: LIBERO's native `info["is_success"]` flag. An episode is labeled successful if the final step's success flag is `True`.
- **Checkpoint used**: `/openpi-libero-2000` (gradient step 2,000 of 30,000 total), chosen as the first checkpoint exhibiting nonzero success across most tasks.
- **Typical success/failure split**: The early checkpoint yields 20–60% success across LIBERO-10 tasks. Eligibility for contrastive fitting follows the activation-level criteria stated in App. F.3.
- **Pre-collected dataset**: Provided as an anonymized HuggingFace dataset (link in the supplementary material).

K.4 ROBOCASA (PI0.5, GR00T N1.5 AND DIFFUSION POLICY)

- **Tasks**: 7 atomic-seen tasks (`CloseFridge`, `CoffeeSetupMug`, `OpenDrawer`, `OpenStandMixerHead`, `PickPlaceCounterToCabinet`, `PickPlaceCounterToStove`, `TurnOnElectricKettle`).
- **Fitting rollouts**: 15 episodes per task (`-num_episodes 15`). These provide the activations for conceptor construction and hyperparameter selection.
- **Test rollouts**: 30 episodes per task per steering condition, collected with fresh seeds disjoint from the fitting set.
- **Initial conditions**: RoboCasa's randomized object placement and kitchen layout per its default `-split pretrain` distribution.
- **Success labeling**: Environment's native success flag.
- **pi0.5 checkpoint**: `robocasa/robocasa365_checkpoints` $\rightarrow$ `pi05_pretrain_human300/multitask_learning/75000` (fully trained on 365-task RoboCasa). This is the production checkpoint, not an early one.

- **GR00T N1.5 checkpoint**: `robocasa/robocasa365_checkpoints` → `gr00t_n1-5/multitask_learning/checkpoint-120000` (365-task multitask pre-training; reported atomic-seen mean 43.0%).
- **Diffusion Policy checkpoint**: `robocasa/robocasa365_checkpoints` → `diffusion_policy/17.40.09_train_diffusion_transformer_hybrid_pretrain_human300/latest.ckpt`
- **Pre-collected datasets**: Provided as anonymized HuggingFace datasets for pi0.5 and GR00T N1.5 (links in the supplementary material).

K.5 Separation of fitting and evaluation data

Rollouts used for conceptor *fitting* (success/failure activation datasets and hyperparameter selection) are entirely separate from those used for steered *evaluation*. Fitting data consists of 15 rollouts per task, collected once per (model, checkpoint, benchmark) tuple and stored to disk. Evaluation data consists of 30 rollouts per task per steering condition, collected with fresh environment resets that use different seeds, object placements, and initial conditions from the fitting set. No episode identifiers or seeds are shared between the two sets. Hyperparameter configurations (layer ℓ, aperture α, steering strength β, and strategy) are selected by maximizing success rate on the fitting rollouts. The selected configuration is then run on the evaluation rollouts, and only the evaluation success rates are reported in the paper.

L Conceptor Computation Details

L.1 Conceptor construction from activations

Given a set of activation vectors $\{h_1, \ldots, h_N\} \subset \mathbb{R}^d$ (one per inference step, after mean-pooling over the token dimension), the conceptor C_α at aperture α is computed as follows:

1. **Mean-center**: $\bar{h} = \frac{1}{N}\sum_i h_i, \quad \tilde{h}_i = h_i - \bar{h}$.
2. **Correlation matrix**: $R = \frac{1}{N}\sum_i \tilde{h}_i \tilde{h}_i^\top = \frac{1}{N}\tilde{H}^\top \tilde{H}$, where $\tilde{H} \in \mathbb{R}^{N \times d}$.
3. **Conceptor**: $C_\alpha = R\,(R + \alpha^{-2} I)^{-1}$.

The contrastive conceptor ("success AND NOT failure") is computed via the Boolean AND-NOT operation defined in Section 3:

$$C_{\text{contr}} = C_s \wedge \neg C_f = \left(C_s^{-1} + (I - C_f)^{-1} - I\right)^{-1}, \tag{9}$$

where C_s and C_f are the success and failure conceptors at the same layer and aperture α. The result is a valid conceptor (symmetric, positive semi-definite, with eigenvalues in $[0, 1]$). For reference, the general Boolean AND between two arbitrary conceptors A and B is:

$$A \wedge B = \left(A^{-1} + B^{-1} - I\right)^{-1}. \tag{10}$$

This form is the canonical conceptor algebra of Jaeger (2014), derived from De Morgan's law $A \wedge B = \neg(\neg A \vee \neg B)$ applied to the OR operation $C_1 \vee C_2 = (R_1 + R_2)(R_1 + R_2 + \alpha^{-2} I)^{-1}$. In implementation, we use the Moore–Penrose pseudoinverse $(\cdot)^\dagger$ to handle near-singular conceptors that arise at small apertures.

Pseudocode.

```
def compute_conceptor(X, alpha):
    """X: (N, d) float64 activation matrix."""
    Xc = X - X.mean(axis=0)           # mean-center
    R  = (Xc.T @ Xc) / max(1, N)      # correlation matrix
    C  = R @ inv(R + alpha**(-2) * I) # conceptor
    return C

def contrastive_conceptor(X_succ, X_fail, alpha):
```

```
    C_s = compute_conceptor(X_succ, alpha)
    C_f = compute_conceptor(X_fail, alpha)
    not_C_f = I - C_f
    # Boolean AND-NOT via canonical form:
    # C_s AND (NOT C_f) = (C_s^-1 + (I - C_f)^-1 - I)^-1
    # pseudoinverse for stability with near-singular conceptors
    C_contr = pinv(pinv(C_s) + pinv(not_C_f) - I)
    return C_s, C_f, C_contr
```

L.2 Steering application

At inference time, the steering transformation is applied as a forward hook on the selected action-expert layer's output:

$$h' = h \cdot M^\top, \qquad M = (1-\beta)\, I + \beta\, C_{\text{contr}}. \tag{11}$$

The matrix $M \in \mathbb{R}^{d\times d}$ is pre-computed once (in `float32` on GPU) and applied via a single matrix multiply per denoising step. The hook fires at every denoising step for the "global" strategy, or uses a different C_{contr} per step for the "per-step" strategy.

Because C_{contr} is symmetric positive semi-definite with eigenvalues in $[0, 1]$, the gate M is a convex combination of the identity and C_{contr} and therefore has spectral norm at most one: it can only attenuate directions of the residual stream and never amplifies activations. This bounds the magnitude of the intervention and is one of the safeguards discussed in the Ethics statement.

L.3 Runtime overhead

- **Conceptor construction** (one-time cost per task): Dominated by the $d \times d$ matrix inverse. For pi0.5 (d=1024), this takes <1 second per (task, layer, α) combination on a single CPU core. For GR00T N1.5 (d=1536), it takes ∼2–3 seconds.
- **Full NPZ construction**: Building conceptors for all tasks, layers, and alphas takes ∼5–15 minutes depending on the number of layers (4 for pi0.5 at selected layers, 16 for GR00T at all layers).
- **Per-step steering overhead**: The matrix multiply $h \cdot M^\top$ for B=1, S=10, d=1024 adds <0.1 ms per denoising step on an NVIDIA A100, negligible compared to the ∼50 ms transformer forward pass. Total inference overhead is <1% wall-clock time.

L.4 Numerical precision

- All correlation matrices and matrix inverses are computed in `float64` (NumPy's default) to avoid numerical issues in the $(R + \alpha^{-2} I)^{-1}$ solve.
- The resulting conceptor matrices are stored as `float32` in compressed `.npz` files.
- At inference time, the steering matrix M is loaded as `float32` and cast to the model's working dtype (`bfloat16`) just before the matrix multiply.
- No eigenvalue truncation is applied. The full $d \times d$ conceptor matrix is stored and used. For diagnostic visualization, we compute eigenvalues via SVD of the mean-centered activation matrix (σ_j^2 from $\tilde{H} = U \Sigma V^\top$), then derive conceptor eigenvalues as $\gamma_j = \sigma_j^2 / (\sigma_j^2 + \alpha^{-2})$.
- Boolean AND is computed via the canonical form $(A^{-1} + B^{-1} - I)^{-1}$ using the Moore–Penrose pseudoinverse to handle near-singular conceptors at small apertures. We observed no numerical issues across all (α, layer) combinations in our sweep grid.

L.5 Memory and storage

- **Conceptor matrix size**: $d^2 \times 4$ bytes (float32). For d=1024: 4 MB per matrix. For d=1536: 9 MB.
- **Full NPZ size**: The LIBERO conceptor NPZ (10 tasks × 4 layers × 5 alphas × 3 matrices + per-step variants) is ∼200 MB compressed. The GR00T NPZ (7 tasks × 16 layers × 5–10 alphas) is ∼1 GB.

- **GPU memory**: At inference time, only one $d \times d$ matrix M is held on GPU per steered layer, adding at most 4 MB for pi0.5 or 9 MB for GR00T. This is negligible compared to the $\sim$3 GB model weights.

M Checkpoint and Training Details

M.1 pi0.5 on MetaWorld

- **Base pretrained checkpoint**: `gs://openpi-assets/checkpoints/pi05_base/params` (Physical Intelligence's released pi0.5 base weights).
- **Fine-tuning dataset**: `/metaworld_ml45` (LeRobot format).
- **Config name**: `pi05_metaworld`.
- **Optimizer**: AdamW with gradient clipping (max norm 1.0).
- **Learning rate**: Cosine decay schedule with 1,000 warmup steps, peak LR 5×10^{-5}, decaying to 5×10^{-6} over 29,000 decay steps.
- **Batch size**: 128.
- **EMA decay**: 0.999.
- **Total gradient steps**: 30,000.
- **Action horizon**: 32.
- **Checkpoint used for activation collection**: Step 5,000 (`/openpi-metaworld-5000`), an early checkpoint chosen because it exhibits mixed success/failure outcomes necessary for contrastive conceptor construction.
- **Fully trained checkpoint**: Step 25,000 (`/openpi-metaworld-25000`).

M.2 pi0.5 on LIBERO-10

- **Base pretrained checkpoint**: `gs://openpi-assets/checkpoints/pi05_base/params`.
- **Fine-tuning dataset**: `physical-intelligence/libero` (LeRobot format).
- **Config name**: `pi05_libero`.
- **Optimizer**: AdamW with gradient clipping (max norm 1.0).
- **Learning rate**: Cosine decay schedule with 10,000 warmup steps, peak LR 5×10^{-5}, decaying to 5×10^{-5} (constant after warmup) over 1,000,000 decay steps.
- **Batch size**: 256 across 4 FSDP devices.
- **EMA decay**: 0.999.
- **Total gradient steps**: 30,000.
- **Action horizon**: 10.
- **Checkpoint used for activation collection**: Step 2,000 (`/openpi-libero-2000`), the earliest checkpoint exhibiting nonzero success across most LIBERO-10 tasks.
- **Released checkpoints**: Steps 2,000, 3,000, and 9,000 (`/openpi-libero-{2000,3000,9000}`).

M.3 pi0.5 on RoboCasa

- **Checkpoint**: `robocasa/robocasa365_checkpoints` $\rightarrow$ `pi05_pretrain_human300/multitask_learning/75000`.
- **Config name**: `pi05_robocasa`.
- **Training recipe**: This is the fully trained production checkpoint from the RoboCasa-365 benchmark, trained on 365 tasks using the `pi05_pretrain_human300` multi-task learning recipe at gradient step 75,000.
- **Action horizon**: Config-dependent (RoboCasa standard).

- **Reason for checkpoint choice**: Unlike MetaWorld and LIBERO where early checkpoints provide the mixed outcomes needed for contrastive conceptor fitting, the RoboCasa fully-trained checkpoint already exhibits sufficient failure rates on most atomic-seen tasks (mean 39–43% success) to provide both success and failure episodes.

M.4 GR00T N1.5 on RoboCasa

- **Checkpoint**: `robocasa/robocasa365_checkpoints` → `gr00t_n1-5/multitask_learning/checkpoint-120000`.
- **Architecture**: NVIDIA Isaac GR00T N1.5 (3B parameters), consisting of a SigLIP + T5 vision-language backbone and a 16-layer DiT action head (d=1536).
- **Training recipe**: 365-task multitask pre-training on RoboCasa. Published atomic-seen mean: 43.0%.
- **Action horizon**: 16 action steps. The client uses `replan_steps=5`, so 11 of 16 predicted steps are discarded at each replan.
- **Denoising steps**: 4 (NVIDIA's default configuration).
- **Image resolution**: N1.5 expects 256×256 images. The openpi RoboCasa client sends 224×224 (pi0.5's resolution); the adapter upscales via bilinear interpolation before inference.
- **Cameras**: Three views (agentview_left, agentview_right, eye_in_hand). The client sends all three; if agentview_right is unavailable, the left view is duplicated as a fallback.

N Baseline Implementation Details

N.1 Filtered behavioral cloning baseline (SFT)

As a parametric counterpart to our inference-time activation-steering intervention, we run a Filtered Behavioral Cloning (filtered-BC) supervised fine-tuning (SFT) baseline that consumes a per-task data budget of 30 on-policy rollouts (twice the 15 rollouts used by COAST) but spends it on weight updates rather than on a steering hook. A larger-budget variant (200 rollouts, up to 1,000 steps) is reported in App. B.1. We follow the recipe of Hu et al. (2022); Black et al. (2025) of self-distilling a base policy on its own successful trajectories. The implementation covers $\pi_{0.5}$ on three simulators (MetaWorld, LIBERO, RoboCasa) and is included in the supplementary code.

N.1.1 Pipeline

For every task in the evaluation split we run, in order:

1. **On-policy rollout.** Roll out N=30 episodes with the base $\pi_{0.5}$ checkpoint π_θ on the target task. At every replan step, the policy emits an action chunk of length H (H=32 for MetaWorld, 10 for LIBERO, and 50 for RoboCasa). We log the observation–action-chunk pair $(o_t, a_{t:t+H})$ that produced each chunk.
2. **Success filtering.** Keep only the $(o_t, a_{t:t+H})$ pairs from episodes that the env's success detector flagged as successful. All samples from unsuccessful episodes are discarded.
3. **LoRA SFT.** Fine-tune π_θ on the filtered buffer with the standard $\pi_{0.5}$ flow-matching loss (Sec. 3) using a LoRA adapter parameterization (Sec. N.1.2).
4. **Adapter merge.** Fold the trained LoRA matrices back into the dense base weights to obtain a single merged checkpoint $\pi_{\theta'}$. The merged model has identical inference cost to π_θ.
5. **Evaluation.** Roll out 30 held-out episodes (different seeds) of $\pi_{\theta'}$ on the same task and report success rate.

The procedure is repeated independently per task: each task gets its own LoRA fine-tune and its own merged checkpoint. This gives filtered-BC a strict per-task advantage over our steering intervention, which uses a single set of activations rather than per-task gradient updates.

N.1.2 TRAINING OBJECTIVE

Let $\mathcal{D}_\tau = \{(o_i, a_i)\}_{i=1}^{M_\tau}$ be the filtered buffer for task τ, where each $a_i \in \mathbb{R}^{H \times d_a}$ is an action chunk. With θ the base $\pi_{0.5}$ parameters and $\Delta\theta$ the LoRA-parameterized update, we minimize the $\pi_{0.5}$ flow-matching loss

$$\mathcal{L}_{\text{FM}}(\Delta\theta;\, \mathcal{D}_\tau) \;=\; \mathbb{E}_{(o,a)\sim\mathcal{D}_\tau,\, t\sim\mathcal{U}(0,1),\, \epsilon\sim\mathcal{N}(0,I)} \left[\left\| v_{\theta+\Delta\theta}(o, a_t, t) - (a - \epsilon) \right\|_2^2 \right], \tag{12}$$

where $a_t = t \cdot a + (1 - t) \cdot \epsilon$ is the noisy action chunk along the flow path and $v_{\theta+\Delta\theta}$ is the velocity field predicted by the LoRA-adapted policy. Only $\Delta\theta$ (and a small set of action/time projection heads, see Sec. N.1.3) receives gradients.

N.1.3 WHAT IS TRAINABLE

The LoRA configuration follows the standard openpi recipe:

- **LoRA on PaliGemma (LM backbone).** Rank-16 adapters with α=16 on every attention and feed-forward projection.
- **LoRA on the $\pi_{0.5}$ action expert.** Rank-32 adapters with α=32 on every attention and feed-forward projection of the 300M-parameter action expert.
- **Action / time projection heads.** The dense `action_in_proj`, `action_out_proj`, `time_mlp_in`, and `time_mlp_out` layers are fully trainable.
- **Frozen.** PaliGemma dense LM weights, action-expert dense weights, and the SigLIP vision tower. The freeze filter is the union of the model's standard LoRA freeze filter and a regex `.*img.*` that catches all SigLIP parameters.

The vision tower is frozen because, at the small per-task data budget ($\sim$150 samples for MetaWorld, $\sim$600 for LIBERO/RoboCasa), leaving SigLIP trainable was the dominant source of catastrophic post-merge regression in our preliminary sweeps (*e.g.* `coffee-push-v3` dropped from 83% rollout success to 0% post-merge eval).

N.1.4 HYPERPARAMETERS

Table 35 lists the hyperparameters used for all three simulators. They are deliberately shared: in early ablations the post-merge eval cliff was step-driven (Adam momentum / gradient-norm growth past $\sim$300 steps), not epoch-driven, so the more sample-rich envs reuse the MetaWorld schedule for safety rather than training longer.

Hyperparameter	Value
Optimizer	AdamW (β_1=0.9, β_2=0.95), grad-norm clip 1.0
LR schedule	Cosine, warmup 20 steps, peak 5×10^{-5}, decay to 5×10^{-6} over 200 steps
Training steps	200
Batch size	8
EMA	disabled
Flow-matching timestep	$t \sim \mathcal{U}(0, 1)$ (default $\pi_{0.5}$)
Per-task rollout budget	30 episodes
Replan stride	10 (MetaWorld), 5 (LIBERO, RoboCasa)
Eval episodes	30 (held-out seeds)

Table 35: Filtered-BC hyperparameters. Identical across MetaWorld, LIBERO, and RoboCasa.

N.1.5 IMPLEMENTATION NOTES

The MetaWorld variant runs entirely in-process: the base `Policy` is loaded once, rollout and evaluation share an `AsyncVectorEnv`, and the merged model is rebuilt as a PyTorch module on GPU for evaluation. The LIBERO and RoboCasa variants use a server–client split because their env libraries require incompatible Python versions (3.8 and 3.11 respectively): per task we spawn a $\pi_{0.5}$ policy server, run rollouts via a WebSocket client in the env's own venv, train + merge in JAX, write the merged checkpoint to a scratch directory, then spawn a fresh server for evaluation. The same

merged-checkpoint contract makes the two execution modes interchangeable from the orchestrator's perspective.

If the success filter yields zero positive samples for a task (the base policy never solves it within the rollout budget), that task is recorded as `skipped_no_successes` and excluded from the aggregate success-rate numbers reported in Sec. 4.1 for the filtered-BC column.

N.2 Linear Contrastive Activation Addition (CAA)

We re-implement the contrastive activation-addition baseline of Rimsky et al. (2024). For each task, we collect residual-stream activations from a fixed set of rollouts and partition them by terminal episode outcome into a success set $\mathcal{H}^+$ and a failure set $\mathcal{H}^-$. The steering direction is the unit-norm difference of class means,

$$\mathbf{v} \;=\; \frac{\bar{\mathbf{h}}^+ - \bar{\mathbf{h}}^-}{\|\bar{\mathbf{h}}^+ - \bar{\mathbf{h}}^-\|_2}, \qquad \bar{\mathbf{h}}^{\pm} \;=\; \frac{1}{|\mathcal{H}^{\pm}|} \sum_{\mathbf{h} \in \mathcal{H}^{\pm}} \mathbf{h},$$

and it is injected at every forward pass through the chosen intervention layer via $\mathbf{h} \leftarrow \mathbf{h} + \alpha\,\mathbf{v}$. The two policy families differ only in where activations are pooled.

- **Flow-matching policy ($\pi_{0.5}$).** We capture the residual stream of the action expert at layer index 11 for every action token of every denoising step. Within each rollout we mean-pool across the action-token axis, yielding one 1024-d vector per (episode, denoising step). The class means $\bar{\mathbf{h}}^{\pm}$ are computed over all such vectors. We also fit per-denoising-step variants $\mathbf{v}^{(t)}$ for $t \in \{0, \ldots, 9\}$ from the matching subset, but report only the single-vector variant in the main results.
- **Autoregressive policy (π_0-FAST).** We capture the pre-LM-head hidden state (2048-d) at every generated action token. There is no denoising-step axis, so each token contributes one sample, and $\bar{\mathbf{h}}^{\pm}$ are computed by pooling all tokens in each class.

A task is eligible only if it has at least three episodes per class; tasks with a single observed outcome are skipped. Steering vectors are computed offline and stored once per task, then loaded by the eval-time hook so that no model weights are modified.

Hyperparameter sweep. We fix the intervention layer to layer 11 for $\pi_{0.5}$, the best-performing captured layer in our preliminary screen, and to the unique pre-LM-head intervention point for π_0-FAST. The only swept hyperparameter is the steering strength $\alpha \in \{0.25, 0.5, 0.75, 1.0\}$. Because $\mathbf{v}$ is unit-normalized, α has the same effective scale across tasks. We evaluate each (task, α) condition on 30 episodes with the same seeds as the unsteered baseline, and select the per-task best α post hoc on this evaluation set. This post-hoc selection favors the baseline relative to COAST, which selects its configuration on the 15 fitting rollouts only; App. B.2 reports a re-run of CAA under COAST's exact train/test protocol with a wider α grid.

N.3 Sparse-Autoencoder Steering (SAE)

As a more expressive contrastive baseline, we replace the raw residual-stream mean difference with a sparse-feature-space mean difference, mediated by a per-task TopK sparse autoencoder (SAE) (Gao et al., 2024). The class- and feature-selection rule follows Khan et al. (2025): instead of picking a small top-K of features by effect size, we keep *all* features that pass a rarity filter and a bilateral-activity filter, then inject the average decoder-space difference over the surviving features.

TopK SAE. For a residual-stream activation $\mathbf{h} \in \mathbb{R}^d$, the encoder produces a non-negative, k-sparse code via

$$\mathbf{f} \;=\; \mathrm{TopK}\big(\,\mathrm{ReLU}(W_{\text{enc}}(\mathbf{h} - \mathbf{b}_{\text{dec}}) + \mathbf{b}_{\text{enc}})\big), \qquad \hat{\mathbf{h}} \;=\; W_{\text{dec}}^{\top}\mathbf{f} + \mathbf{b}_{\text{dec}},$$

where $\mathrm{TopK}(\cdot)$ retains the k largest entries and zeros the rest. We unit-normalize each decoder column after every optimizer step. The ReLU before the TopK guarantees $\mathbf{f} \geq \mathbf{0}$, an assumption the bilateral filter below relies on. We train one SAE per task on the same captured activations used for CAA, hold out 5% as a reconstruction-quality gate, and drop any task whose hold-out variance-explained falls below 0.80 from the SAE arm of the comparison.

Steering vector via class means in feature space. After training, we encode every captured activation through the frozen SAE and split the resulting codes by episode outcome. For each feature index k we compute

$$\mu_k^+ \;=\; \mathbb{E}_{\mathbf{f}\in\mathcal{F}^+}\big[f_k\big], \quad \mu_k^- \;=\; \mathbb{E}_{\mathbf{f}\in\mathcal{F}^-}\big[f_k\big], \quad r_k \;=\; \frac{\#\{\,\mathbf{f}\in\mathcal{F}^+\cup\mathcal{F}^- : f_k>0\,\}}{|\mathcal{F}^+|+|\mathcal{F}^-|}.$$

We then apply the two feature filters of Khan et al. (2025):

- **Filter A (rarity).** Keep k if $r_k \geq 0.005$. This drops the long dead-feature tail that any TopK SAE produces. Below this fire rate, the estimates of $\mu_k^{\pm}$ are statistically unreliable.
- **Filter B (bilateral activity).** Keep k if $\min(\mu_k^+, \mu_k^-) \,/\, \max(\mu_k^+, \mu_k^-) \;\leq\; 0.5$. Features that fire similarly in both classes carry no contrastive signal.

Let $\mathcal{S}$ denote the surviving feature set. The latent-space contrast is $v_k^{\text{lat}} = (\mu_k^+ - \mu_k^-)\mathbb{1}[k \in \mathcal{S}]$, and the residual-stream steering vector is its image through the decoder, unit-normalized:

$$\tilde{\mathbf{v}}_{\text{SAE}} \;=\; W_{\text{dec}}^{\top}\,\mathbf{v}^{\text{lat}}, \qquad \mathbf{v}_{\text{SAE}} \;=\; \tilde{\mathbf{v}}_{\text{SAE}} \,/\, \|\tilde{\mathbf{v}}_{\text{SAE}}\|_2.$$

At eval time, $\mathbf{v}_{\text{SAE}}$ is injected via the same additive hook as CAA, $\mathbf{h} \leftarrow \mathbf{h} + \alpha\,\mathbf{v}_{\text{SAE}}$, at the same intervention point. Unit-normalization matters because the number of surviving features, and therefore the raw norm of $\tilde{\mathbf{v}}_{\text{SAE}}$, varies substantially across tasks. Without it, identical α would correspond to different effective intervention strengths per task.

Hyperparameters. The SAE width is $d_{\text{SAE}} = 4d$ (4096 for $\pi_{0.5}$, 8192 for π_0-FAST) and the TopK sparsity is $k = 64$. Optimization uses AdamW with learning rate 3×10^{-4}, $\beta = (0.9, 0.999)$, batch size 4096, for 30,000 steps. The encoder is initialized from the unit-normalized decoder transpose. We do not sweep over k, d_{SAE}, or expansion factor: in line with the recipe of Khan et al. (2025), k is set by the rarity filter rather than tuned, and the only knob exposed to the eval-time sweep is the steering strength.

Sweep. We sweep $\alpha \in \{0.25,\, 0.5,\, 1.0,\, 2.0\}$, deliberately wider than the CAA grid because $\mathbf{v}_{\text{SAE}}$ is constructed from a sparse set of decoder columns and may need a different effective magnitude than the dense CAA mean difference. As with CAA, the intervention layer is fixed (layer 11 for $\pi_{0.5}$, pre-LM-head hidden state for π_0-FAST), each condition is evaluated on 30 episodes, and the per-task best α is selected post hoc (see App. B.2 for the matched-protocol re-run). We do not implement feature *clamping*, overwriting f_k to a target value and re-decoding. The additive form above is the simplest faithful instantiation of the Khan et al. (2025) recipe and cleanly compares to CAA, which differs only in the basis used to compute the contrast.

O Extended Comparison of Adaptation Paradigms

Relation to prior conceptor work. The conceptor formalism and its Boolean algebra originate in Jaeger (2014) and were first applied to LLM steering by Postmus & Abreu (2024) and Postmus & Abreu (2025). COAST adapts this machinery to a setting that differs structurally from single-forward-pass language models. Actions in flow-matching VLAs are produced through iterative denoising, during which the discriminative subspace shifts from coarse to fine-grained; the per-step conceptors of Sec. 3.2 track these non-stationary subspaces across the flow, and the multiplicative gate corrects the trajectory at every step without compounding errors, in contrast to the additive steering of prior conceptor work. The success/failure construction is itself a source of findings: it reveals that the outcome-relevant subspace is low-rank but not rank-one (Sec. 4.3), which explains why single-direction methods recover only part of the gain, and that failure geometry is shared across tasks while success geometry is task-specific (Sec. 4.4), which enables transfer with a computable proxy. These empirical observations about how VLAs encode task outcomes hold independently of the particular steering operator.

To contextualize our contribution, Table 36 compares COAST against recent methods for inference-time adaptation and steering of robot policies. Because the computational requirements and operational paradigms of these methods vary significantly, we categorize them along four distinct axes that define their deployment burden. Below are the definitions of each comparison criterion:

Table 36: Comparison of COAST against existing VLA adaptation paradigms. COAST is the only method that achieves holistic, multi-dimensional steering without requiring gradient updates, external models, or computationally expensive test-time search.

Method	Training-Free	Self-Contained	Single-Pass
Fine-Tuning / SFT (Yuan et al., 2024)	×	✓	✓
VLM Search / VLS (Liu et al., 2026)	✓	×	×
Test-Time Search / TACO (Yang et al., 2025)	✓	×	×
SAE Steering (Huben et al., 2024)	×	✓	✓
Activation Addition (Turner et al., 2024)	✓	✓	✓
COAST (Ours)	✓	✓	✓

Training-Free. A method is considered training-free if it does not require gradient updates to the policy weights, auxiliary model weights, or any explicitly learned parameter modules. Approaches such as supervised fine-tuning (SFT) (Yuan et al., 2024; Mitra et al., 2025) and explicitly trained steering interfaces (Chen et al., 2026; Das et al., 2025) require offline retraining or the optimization of auxiliary modules, thus incurring a high computational cost before deployment. Contrastive Representation Regularization (Kim et al., 2025) similarly modifies the core policy weights. Conversely, COAST relies on closed-form Boolean subspace algebra computed directly from rollouts (Postmus & Abreu, 2024), bypassing gradient-based optimization entirely.

Self-Contained. A method is self-contained if it does not rely on querying external foundation models (e.g., Vision-Language Models) or explicit external reward models at inference time. Several recent inference-time adaptation frameworks rely on VLM-based planning, scoring, or verification to guide the robot's actions. For example, VLS (Liu et al., 2026) relies on external VLM-based rewards for steering, while Shah et al. (2025) and Wu et al. (2026) use VLMs for runtime reasoning and action alignment verification. These external dependencies introduce significant latency and point-of-failure risks. COAST is purely internal, operating exclusively on the native residual stream of the frozen VLA. We exclude methods like VLS (Liu et al., 2026) and TACO (Yang et al., 2025) because their reliance on external VLMs or multi-sample search makes their operational cost structurally incomparable to COAST's single-pass, negligible-overhead execution.

Single-Pass. A single-pass method alters the model's behavior without requiring multi-sample search, Monte Carlo sampling, or iterative refinement loops during action generation. Methods that rely on test-time search or trajectory sampling, such as TACO (Yang et al., 2025) or VLM-guided rollout evaluation, multiply the inference latency of the policy by the number of samples required. COAST is applied as a fixed multiplicative gate to the hidden states during the standard forward pass, yielding effectively zero latency overhead.

Subspace Control. This criterion distinguishes methods that intervene on multi-dimensional, outcome-relevant geometric structures from those that intervene on isolated directions. Standard Activation Addition (CAA) (Turner et al., 2024; Rimsky et al., 2024) shifts activations along a single rank-one vector. Similarly, Sparse Autoencoder (SAE) steering (Huben et al., 2024; Buurmeijer et al., 2026; Khan et al., 2025) isolates and scales individual, disentangled features. While effective for localized semantic concepts (Häon et al., 2025), these methods offer only partial, constrained control. Drawing on conceptor theory (Jaeger, 2014; Postmus & Abreu, 2025), COAST leverages a continuous, high-dimensional subspace (Zou et al., 2025) that explicitly separates successful and failed behaviors. This structure enables holistic, multi-dimensional steering rather than feature-by-feature manipulation.

By satisfying all four criteria, COAST uniquely addresses the representation bottleneck in VLA policies without the latency of external VLM queries (Liu et al., 2026), the computational cost of test-time search (Yang et al., 2025), or the partial control of single-feature interventions (Khan et al.,

2025). We view this framing as essential for standardizing how inference-time adaptation methods are evaluated in embodied AI, as matched-cost comparisons (e.g., COAST vs. Activation Addition) yield significantly more rigorous insights into the geometry of policy latent spaces than comparing single-pass internal methods against multi-sample external searches.

P Computational Overhead at Inference Time

A natural concern about COAST is that its multiplicative gate multiplies the residual stream by a $d \times d$ matrix at every denoising step, where additive baselines such as CAA and SAE only add a d-vector. We show that at the dimensionalities current VLA policies operate at, this gap is empirically invisible: COAST's per-call latency is statistically indistinguishable from the additive baselines.

Setup. We benchmark $\pi_{0.5}$ on LIBERO on a single NVIDIA B200, with the forward hook registered on the action expert at layer $L{=}11$, the same intervention point used in our main results. For each of four conditions (no steering, CAA, SAE, COAST), we discard 10 warmup calls and then time $N{=}200$ subsequent `policy.infer(obs)` calls gated by `torch.cuda.synchronize()` on both ends. cuDNN benchmarking is disabled and the same observation is reused across calls so we measure raw overhead rather than data-dependent variance. All steering conditions use the global variant; per-step variants have identical per-call latency since the hook fires the same number of times per `infer()`.

All three steering methods add similar overhead. COAST adds 30.85 ms (median) to a 127.12 ms baseline forward pass. The additive CAA and SAE baselines add 31.47 ms and 30.29 ms respectively. The three steering methods are statistically indistinguishable: all three medians lie within 1.2 ms of each other, well inside the per-condition IQR. The implication is that the overhead reported in Table 37 is not the cost of COAST's matmul; it is the fixed cost of registering and dispatching a PyTorch forward hook, paid identically by every hook-based steering method. Replacing COAST with the cheapest possible additive baseline would not save measurable inference time.

Method	Median (ms)	IQR (ms)	Δ (ms)	Δ (%)
no steering	127.12	1.05	—	—
CAA (global)	158.58	5.42	+31.47	+24.75%
SAE (global)	157.40	3.03	+30.29	+23.82%
COAST (global)	**157.97**	**3.25**	**+30.85**	**+24.27%**

Table 37: **Per-call inference latency on one B200, $\pi_{0.5}$ LIBERO, $L{=}11$.** Median over $N{=}200$ `infer()` calls after 10 warmup calls; cuDNN benchmarking disabled. Each `infer()` runs all 10 denoising steps; the steering hook fires once per step. COAST's median latency is within 1.2 ms of both additive baselines, well inside their IQRs.

Offline cost. The COAST steering artifact is computed once per task offline, in seconds, by a single eigendecomposition of a 1024×1024 activation covariance matrix. CAA reduces to two class-conditional means and is similarly free. SAE is the outlier at training time, approximately one GPU-hour per task to fit the per-task TopK autoencoder, but this cost is amortized across all evaluation runs that use the resulting $\mathbf{v}_{\text{SAE}}$. None of these offline costs are paid at deployment time.