Mechanistic Interpretability for Steering Vision-Language-Action Models

Bear Häon* **Kaylene Stocking*** **Ian Chuang** **Claire Tomlin**
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

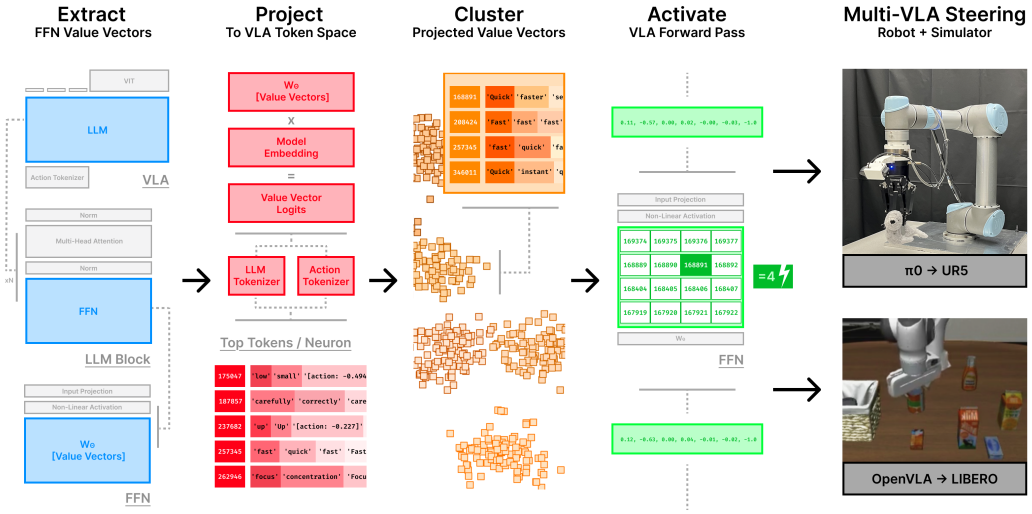


Figure 1: **We present a framework for steering Vision-Language-Action (VLA) models.** We extract FFN vectors, project them to the VLA token space, cluster them by semantic alignment, and inject activations at inference time to modulate behavior. Our experiments demonstrate interpretable zero-shot control in both simulation (OpenVLA in LIBERO) and on a physical robot (π_0 on a UR5).

Abstract: Vision-Language-Action (VLA) models are a promising path to realizing generalist embodied agents that can quickly adapt to new tasks, modalities, and environments. However, methods for interpreting and steering VLAs fall far short of classical robotics pipelines, which are grounded in explicit models of kinematics, dynamics, and control. This lack of mechanistic insight is a central challenge for deploying learned policies in real-world robotics, where robustness and explainability are critical. Motivated by advances in mechanistic interpretability for large language models, **we introduce the first framework for interpreting and steering VLAs via their internal representations, enabling direct intervention in model behavior at inference time.** We project feedforward activations within transformer layers onto the token embedding basis, identifying sparse semantic directions - such as *speed* and *direction* - that are causally linked to action selection. Leveraging these findings, we introduce a general-purpose activation steering method that modulates behavior in real time, without fine-tuning, reward signals, or environment interaction. We evaluate this method on two recent open-source VLAs, π_0 and OpenVLA, and demonstrate zero-shot behavioral control in simulation (LIBERO) and on a physical robot (UR5). **This work demonstrates that interpretable components of embodied VLAs can be systematically harnessed for control—establishing a new paradigm for transparent and steerable foundation models in robotics.**

*Equal contribution. Correspondence to: {bear.haon, kaylene}@berkeley.edu

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

Keywords: Mechanistic Interpretability, Vision-Language-Action Models, Foundation Models for Robotics

1 Introduction

Large language-conditioned models represent a paradigm shift for robotics. By leveraging Internet-scale natural language and image data, these models promise to yield generalist robot policies that can follow directions, exhibit common-sense reasoning about their environments, and quickly adapt to novel tasks and scenarios.

One leading type of frontier robotics model is the Vision-Language-Action (VLA) model, where a pre-trained Vision-Language Model (VLM) is trained to produce robot actions conditioned on a task description and image observations. While most VLAs are designed to be fine-tuned for specific tasks, future models may even be able to perform a variety of tasks without the need for fine-tuning. Indeed, the π_0 model [1] can often complete zero-shot evaluation tasks on the DROID platform [2] and Gemini Robotics demonstrates zero-shot control via code generation [3]. This promise of generality and ease of deployment signals a large shift in how we deploy robot systems. In contrast to rigorous evaluation and certification on individual tasks in factories or warehouses, robots will be placed in novel environments and expected to behave well "out-of-the-box". This raises important questions about safety, robustness, and transparency. How and when can we trust that VLA policies will behave safely? When they fail, how can we diagnose and fix the underlying problems?

A similar set of questions and challenges is being faced by Large Language Model (LLM) researchers. While they have by no means been solved, the field of *mechanistic interpretability* has yielded a number of exciting insights about the inner workings of LLMs. For example, researchers have discovered the functional role of specific attention heads [4] and identified semantically interpretable concepts in model activations [5, 6]. There is also evidence that mechanistic interpretability can be useful for diagnosing problems with models in practice [7]. These insights lead us to ask: can mechanistic interpretability also help with understanding and shaping the behavior of frontier models in robotics?

In this work, we present a series of experiments that together allow us to provide a resounding affirmative answer. We focus our efforts on an interpretability method that analyzes the semantic meanings of vectors in the output weight matrix of the transformer block's feed-forward layer (FFN) [8]. Although this technique has less power to disentangle concepts that might be spread across multiple neurons in the model than alternatives such as sparse autoencoders [5], it allows us to interpret and steer the model without any additional training data. This is a key advantage in robotics where training data is expensive and difficult to acquire, and where available open datasets are small compared to what is available in the natural language domain. Our chosen method leads us to several surprising conclusions about VLAs, including:

1. Despite being trained to produce only robot actions, the internal activity of the VLA model is largely semantic, with less than 25% of FFN neurons being rewired for action prediction and a large fraction of the remainder representing clear, semantically interpretable concepts.
2. Internal model concepts such as "slow" are causally linked to robot actions that express these concepts (i.e. moving the end effector more slowly), despite the model being trained without any feedback on the meaning or consequences of different action tokens.
3. Targeted interventions on model concepts allow VLA behavior to be modified at inference time out-of-the-box, without any fine-tuning data or environment interaction required to select the intervention.

These contributions represent the first attempt to use mechanistic interpretability techniques developed for LLMs with large robotics models. Our exciting initial results speak to the great promise of adapting these techniques to the robotics context, where there is an especially strong need for AI models that are safe and transparent.

2 Related Work

Mechanistic Interpretability and Robotics. The size and complexity of deep learning systems make it challenging for humans to understand how they arrive at a particular output from a given input. To help address this problem, a variety of methods have been proposed to help expose and interpret the internal representations and computations in models. The most powerful form of interpretability work doesn't simply yield qualitative insights, but elucidates the causal mechanisms that determine the output of the model - this is often referred to as "mechanistic interpretability". Despite promising developments in mechanistic interpretability in the vision [9] and natural language [5, 10, 11] settings, there has been relatively little work investigating DNN interpretability for robotics. In [12], they aim to interpret a deep learned policy by associating its factors of variation with logic rules, but the system they investigate is small in scale. In [13], they use techniques inspired by vision interpretability work to improve a robotic image-condititioned policy. Finally, there has been recent interest in interpretability for reinforcement learning policies, as surveyed in [14].

3 Interpreting VLAs

Deep neural networks build their predictions layer by layer, with features in one layer supporting further computations and representations in the next. According to the "linear representation hypothesis", intermediate network layers often represent different features or concepts as directions in the latent embedding space [15]. This means that if, for example, a model represents the concept of "fast", if we can identify the corresponding direction in its latent space, we can understand when and how the model uses this concept to produce its final output. Serendipitously, it turns out that for many large models, semantically meaningful directions align with individual neurons. This fact has been used to help interpret both vision [9] and language [16, 8] models. Recently, there has been excitement around using sparse autoencoders (SAEs) to uncover even more meaningful directions [5]. However, training SAEs require large amounts of data, and the results vary depending on the exact dataset used. Both of these factors are significant barriers in the robotics context, where open-source datasets are limited and are unlikely to cover all of the semantic concepts that the model learned and might retain from VLM pretraining. Therefore, in our work, we focus on single-neuron features.

Specifically, we follow the approach outlined in [8] and focus on the feedforward layer of each transformer block:

$$\text{FFN}(x) = f_\theta(x)^T W_\theta \tag{1}$$

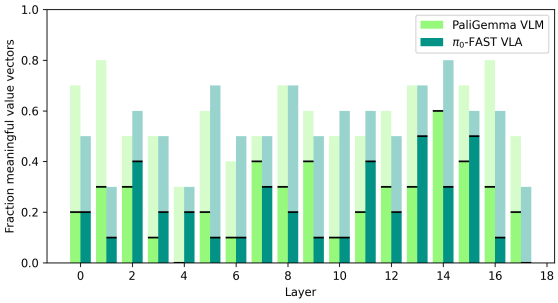
where $x \in \mathbb{R}^n$ is the input to the layer, $f_\theta(x) \in \mathbb{R}^m$ is an input-dependent set of activations, and $W_\theta \in \mathbb{R}^{m \times n}$ is an input-independent parameter matrix. Using $w_\theta^{(i)} \in \mathbb{R}^n$ to refer to the ith row of W_θ, Equation (1) can be rewritten as follows:

$$\text{FFN}(x) = \sum_i [f_\theta(x)]_i w_\theta^{(i)} \tag{2}$$

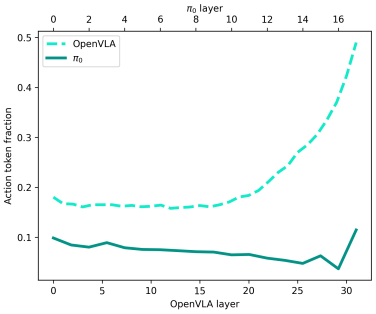
This allows us to interpret the FFN as a weighted sum over the $w_\theta^{(i)}$ vectors, which we refer to as *value vectors*. Since these vectors are independent of the inputs, they can be thought of as basis functions for the output of the FFN. Furthermore, as they belong to the same linear space as the final output of the transformer, they can be interpreted as probability distributions over possible output tokens. Therefore, we can assign semantic meanings to value vectors based on the set of tokens to which they assign the highest probability.

3.1 How does VLA Training Affect Concepts Learned During VLM Pre-training?

During VLA training, a pre-trained VLM is fine-tuned to imitate the control actions of expert robot trajectories conditioned on a task description and image and/or state observation, using a dataset such as Open X-Embodiment [17]. The most common strategy for handling the actions, followed by VLAs such as RT-2 [18], OpenVLA [19], and π_0-FAST [2], is to denote a small set of rarely used VLM tokens as "action tokens" to be decoded into control outputs. The final output of the



(a) **Meaningful patterns in top value vector tokens**. VLA training does not substantially change the proportion of FFN value vectors which have interpretable patterns (top lighter bars) and semantically meaningful patterns (bold bottom bars) in their top tokens.



(b) **Action tokens are incorporated into every layer of the VLA.** However, they make up the largest proportion of final layer value vectors.

Figure 2: VLA training incorporates action tokens into FFN value vectors but retains semantic meanings from VLM pre-training.

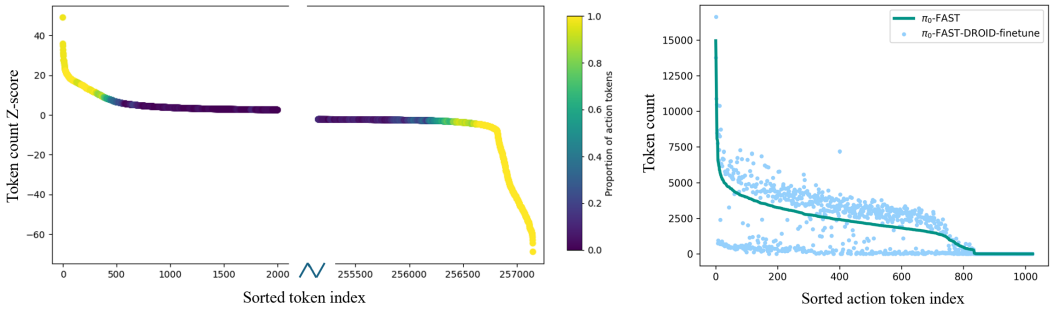
model during VLA training is exclusively action tokens, which raises the question: does the VLA still use natural language-like representations to choose appropriate control actions? If so, how do language and action interact in the trained model? Our chosen interpretability technique allows us some insight into these questions:

VLA Value Vectors Retain Semantic Meanings. First, we aimed to quantify whether the VLM value vectors represent semantically interpretable concepts, and if so, whether this is preserved after VLA training. We focused on π_0 and its base pretrained VLM, PaliGemma-3B [20], because its larger vocabulary size, compared to OpenVLA's Llama2-7B base VLM [21], makes individual tokens easier to interpret. We randomly selected 10 value vectors from each layer of the VLM and VLA models and examined whether at least 4 of the top 30 tokens followed a common pattern (adopting the methodology from [8]. The results are shown in Figure 2a. Value vectors throughout both the VLM and VLA models exhibit identifiable patterns, many of them semantic, at similar rates. Therefore, VLA training does not cause FFN outputs to lose meaningful organization of concepts, despite producing only non-semantic tokens at the final output. We show qualitative examples of value vectors in the Appendix.

Action Tokens Appear in All VLA Layers. We show the proportion of top-100 tokens which are action tokens for value vectors in each layer of the model (Figure 2b). For both OpenVLA and π_0, action tokens are most common at the final layer, but make up at least a few percent of every layer. This suggests that there is not a hard transition from "thinking about the task" in earlier layers to "thinking about control" in later layers. Instead, it appears VLA models learn to reason about and refine their control action predictions continuously, starting from their earliest computations.

3.2 How Task-Specific Fine-Tuning Affects VLA Concepts

After VLA pre-training, models generally need to be fine-tuned for a specific robot setup and set of tasks to realize good performance. To understand what happens to the value vectors during fine-tuning, we compared the base π_0-FAST model with a checkpoint fine-tuned on the DROID dataset [22]. We compared top tokens in the value vectors using a two-proportion z-test statistic to quantify the significance of an increase (or decrease) in the number of occurrences of each token between the two models. Our results are shown in Figure 3a and 3b. We find that action tokens account almost exclusively for the most significant differences between the two sets of value vectors, and that this shift occurs because the fine-tuned model develops a more uneven, specialized distribution across action tokens compared to the relatively flat distribution of the pre-trained model. There is also a modest boost to some of the most common tokens in the DROID task instructions: these tokens appear 1.2 times more frequently in the fine-tuned model value vectors than in the pre-trained ones (see the Appendix for more details). Overall, these results suggest that the main effect of fine-tuning



(a) **Task fine-tuning mainly affects action tokens**. The most up-weighted and down-weighted tokens between the π_0-FAST and π_0-FAST-DROID-finetune models are action tokens.

(b) **Fine-tuning induces a more specialized (less general) distribution of action tokens across value vectors.**

Figure 3: Task fine-tuning mainly affects action tokens in FFN value vectors.

is to re-wire the model to more easily produce action tokens utilized in the fine-tuning tasks and neglect the others, with only a modest effect on semantic tokens.

3.3 Summary

The above results suggest that VLAs reason about control actions by mixing semantic concepts from pre-training with action tokens at all layers. However, these results do not prove a direct causal link between specific semantic concepts and robot behavior. If such a link exists, we expect to be able to *intervene* on semantic concepts to change behavior. In the following section, we examine whether the semantic concepts we uncovered can support such interventions.

4 Steering VLAs

We formalize our method for *interpretable activation-level steering* of VLAs (Figure 1). This technique modifies selected neurons within the FFN submodules of transformer blocks inside the VLA's base VLM, steering robot behavior in real time - without fine-tuning or environment reward signal.

Let $x \in \mathbb{R}^n$ be the residual input to a transformer FFN. As shown in Equation (2), the FFN output is a sum of fixed value vectors $w_\theta^{(i)} \in \mathbb{R}^n$ weighted by input-dependent activations $f_\theta(x) \in \mathbb{R}^m$. Motivated by evidence that individual FFN neurons encode semantic features [16, 5], we override a subset $\mathcal{S} \subseteq \{1, \ldots, m\}$ of activations using a fixed scalar $\alpha \in \mathbb{R}$. Each $\mathcal{S}$ corresponds to an interpretable neuron cluster aligned with a control concept - e.g., *fast*, *up*, *careful* - identified by grouping neurons with similar token projections (Figure 1, steps 1–3), either manually or via kNN over semantic embeddings.

$$\tilde{f}_\theta^{(i)}(x) = \begin{cases} \alpha & \text{if } i \in \mathcal{S} \\ [f_\theta(x)]_i & \text{otherwise} \end{cases} \tag{3}$$

$$\text{FFN}_{\text{steered}}(x) = \sum_{i=1}^{m} \tilde{f}_\theta^{(i)}(x) \cdot w_\theta^{(i)} \tag{4}$$

This induces a residual shift $\Delta x = \text{FFN}_{\text{steered}}(x) - \text{FFN}(x)$ that propagates through the transformer and modulates the final VLA action token distribution.

Implementation. In OpenVLA (PyTorch), we apply a forward hook on the FFN's `down_proj` to overwrite neuron activations. In π_0 (JAX), the neuron indices $\mathcal{S}$ and activation coefficient α are passed into a refactored FFN. Both realize the same operator $\mathcal{I}_\mathcal{S}^\alpha(f_\theta(x))$ for real-time robot control.

4.1 Simulation Experiments

We evaluate our VLA steering method within the suite of ten long-horizon manipulation tasks making up the LIBERO-Long benchmark [23], visualized in Figure 4. Rollouts are seeded determin-

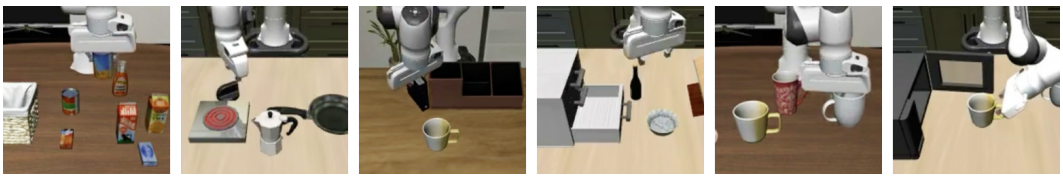
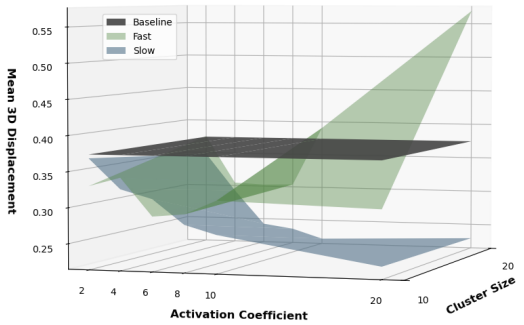
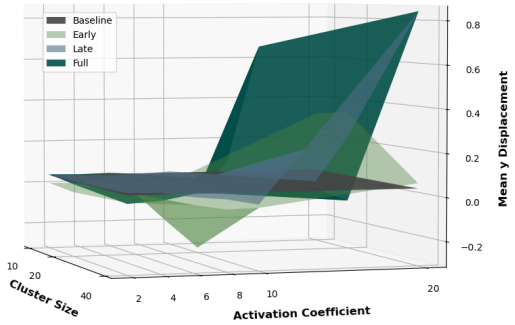


Figure 4: **Sample tasks from LIBERO-Long.** Six representative long-horizon tasks - involving sequential manipulation goals such as object placement, containment, and appliance interaction.



(a) **Steering motion magnitude interventions.** The effect of fast and slow cluster interventions across cluster sizes and activation coefficients. Fast clusters consistently lead to larger end-effector displacement.



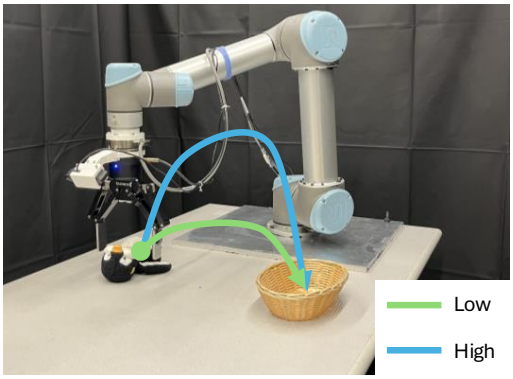
(b) **Temporal localization interventions.** Mean Y-displacement for *up*-cluster activations injected at early, late, and full model depths. Full clusters produce the largest average motion effects.

Figure 5: **Simulation results:** Steering OpenVLA with value vector interventions.

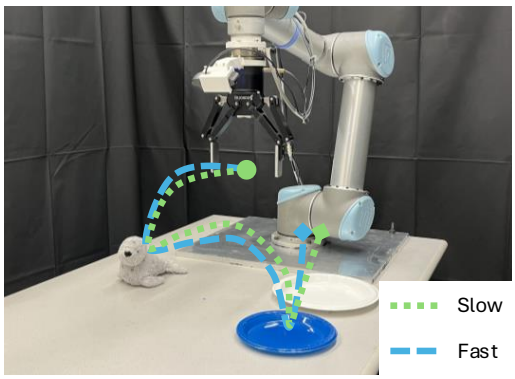
istically for reproducibility, enabling precise comparisons between baseline and intervention trajectories. All simulation experiments use the 7B-parameter OpenVLA model, with the fine-tuned LIBERO-Long checkpoint, implemented in PyTorch and executed on a NVIDIA H100 GPU.

Steering Motion Interventions. To evaluate whether interpretable semantic structure in VLA value vectors can steer physical behavior, we construct *fast* and *slow* aligned clusters by manually selecting value vectors whose top-projected tokens reflect motion magnitude semantics. These clusters are injected into the model's residual stream with a fixed coefficient, sweeping over cluster sizes $[10, 20]$ and activation strengths $[2, 4, 6, 8, 10, 20]$. Each configuration is evaluated over 10 long-horizon tasks, using 10 rollouts per task. The fast clusters consistently induce larger end-effector displacements across tasks, with an average improvement of 27.73% over slow clusters and maximum gains of 148.54% in some configurations. All 10 matched comparisons yield statistically significant differences ($p < 0.001$, paired t-test), and effect sizes ($d = -0.091$ – 1.419) suggest consistent directional influence. These results demonstrate that directional behavior can be modulated by activating semantically interpretable value vectors alone (Figure 5a).

Temporal Localization Interventions. We examine how the position of activated value vectors within the VLA affects steering performance. Using kNN clustering over token-projected value vector embeddings, we extract *up*-themed clusters and restrict their injection to early, late, or all layers. Each configuration spans three cluster sizes and six activation coefficients, with all other variables held constant. Across conditions, full-layer interventions yielded the largest average Y-displacements of the end-effector ($\mu = 0.098$), followed by late-layer ($\mu = 0.086$) and early-layer ($\mu = 0.007$) injections. Effect-size contrasts were most pronounced between early and late depths ($d = -0.376$), smaller between early and full ($d = -0.321$), and negligible between late and full ($d = -0.062$). As shown in Figure 5b, full-layer activations have the strongest effect on average, yet late-layer interventions can match them at higher intensities and larger cluster sizes. This is consistent with the expectation of explicit motion semantics concentrating in the model's final stages.



(a) **Low/high transport.** The robot picks up a toy penguin and places it into a basket, with variations in the robot's trajectory height during data collection.



(b) **Slow/fast transport.** The robot picks up a toy seal and places it onto a plate, with variations in the robot's movement speed during data collection.

Figure 6: **Physical robot experiments:** Steering π_0 on a UR5.

4.2 Robot Experiments

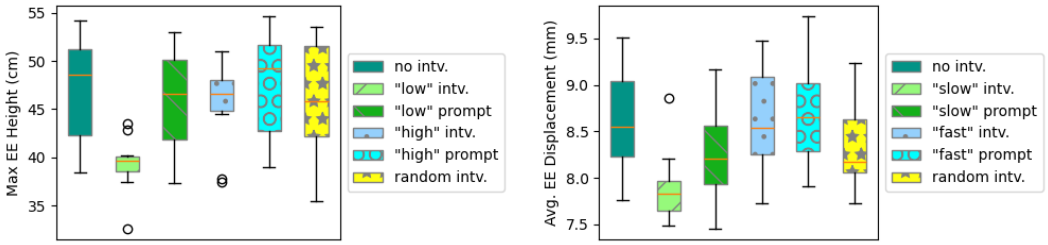
We also evaluate our VLA steering method using the 3B-parameter π_0-FAST VLA on a real robot. Specifically, we test the ability to steer binary control opposites - *slow vs. fast* and *low vs. high* - in two pick-and-place tasks with a UR5 robot arm. We use a JAX implementation of π_0-FAST running on a NVIDIA A4500 GPU. Details on our custom robot setup are provided in the Appendix.

Fine-Tuning. Unlike benchmarks such as DROID, which use a standardized robot platform and include fine-tuned π_0-FAST checkpoints, our robot setup is not represented in the π_0-FAST training data and cannot reasonably perform tasks zero-shot. To enable meaningful evaluation, we fine-tune with LoRA [24] on small datasets collected from our robot. Although fine-tuning on a limited dataset may hinder generalization and steering capability, it enables controlled experimentation: we manage the full data collection process and can introduce variations and behaviors in the robot data that we wish to steer during evaluation. Crucially, we do not explicitly label these variations in the VLA prompt during training, allowing us to test whether the VLA can reproduce these specific behaviors through steering by associating them with high-level semantic concepts.

Robot Tasks. We evaluate steering on two tasks. The first task, *Low/High Transport*, involves the robot picking up a toy penguin and placing it in a basket and includes 75 episodes with unlabeled variations in how high the penguin is lifted during transport. The second task, *Slow/Fast Transport*, involves the robot picking up a toy seal and placing it on a specified plate and includes 120 episodes with variations in the speed of the robot's motion. Visualizations of both tasks are shown in Figures 6a and 6b, and data collection and fine-tuning details are provided in the Appendix.

For both tasks, we evaluate steering interventions where we hand-select and upweight a cluster of semantically meaningful vectors. We create a *low* and *high* themed cluster for *Low/High Transport* and *slow* and *fast* themed cluster for *Slow/Fast Transport*. We also compare these interventions against several baselines: (1) running the model without any intervention, (2) adding qualitative descriptors to the prompt (e.g. appending *low* to the beginning of the prompt), and (3) applying steering with randomly upweighted vectors. These comparisons evaluate the model's default behavior without intervention, whether prompt modification alone can influence behavior, and whether selecting steering vectors based on semantic concepts are more effective than simply selecting random vectors. Specific details on each steering intervention and baseline is in the Appendix.

Task Evaluation. For *Low/High Transport*, we perform 10 independent rollouts for each steering variant and baseline and analyze the maximum height of the end-effector relative to the robot's base over the course of the task. We then record the maximum height across 10 rollouts for each variant in Figure 7a. In *Slow/Fast Transport*, we focus on measuring the robot's speed. To do this in a consistent manner we run inference for all steering variants and baselines simultaneously, but only execute the baseline predicted action with no intervention. We measure average end-effector dis-



(a) **Low/high transport.** Box plots for maximum end-effector height (cm) showing distribution across 10 rollouts for each steering intervention (intv.) and baseline.

(b) **Slow/fast transport.** Box plots for average end-effector displacement (mm) between each successive action showing distribution across 10 rollouts for each steering intervention (intv.) and baseline.

Figure 7: **Physical robot experiments:** Binary control opposites.

placement between each step along the trajectory across 10 rollouts for each variant and plot results in Figure 7b. More detailed analysis of individual trajectories for both tasks are in the Appendix.

Robot Results. Our results suggest that steering interventions can meaningfully influence the actions of π_0-FAST. In the *Low/High Transport* task, the *low* intervention generated the lowest overall trajectories. Similarly, the *slow* intervention in *Slow/Fast Transport* resulted in the slowest overall movements. However, while *low* and *slow* interventions clearly altered behavior, their opposites (*high* and *fast*) resembled the baseline without intervention. This may be because the model already "considers" the baseline trajectory to be fast and high. Nevertheless, the minimal difference between using a random intervention compared to no intervention suggests that choosing semantically meaningful vectors can more effectively guide the model towards desired behavior. Finally, we found that steering interventions were more impactful than simply adding target descriptive words to the prompt. In both tasks, modifying the prompt had weaker impact than our steering intervention.

5 Discussion and Conclusion

This work shows that VLAs can be steered via interpretable internal components. By activating sparse, semantically meaningful neurons in FFN layers—aligned with concepts like "up" and "slow"—we modulate robot behavior at inference time. Experiments demonstrate that our method generalizes across models and tasks, influencing both simulated and real-world behavior, including long-horizon LIBERO tasks and binary control opposites on a UR5.

From a mechanistic perspective, this work provides causal evidence that VLAs retain structured, compositional semantics even after fine-tuning. That activating a small set of semantically aligned neurons can shift motion behavior suggests control-relevant features remain accessible and manipulable. These effects emerge without explicit behavioral supervision, indicating that VLAs internalize semantic concepts during pretraining in a way that supports compositional reasoning across layers.

For robotics practitioners, this method offers a simple, interpretable tool for post-deployment behavior shaping. It requires no retraining, runs on existing policies, and enables fast testing and adjustment. Our experiments show that it can modulate some aspects of robot behavior—such as end-effector speed or height—and suggest that this modulation may be more effective than using modified prompts.

Looking ahead, activation steering opens a new interpretable control interface for guiding VLA behavior. As embodied foundation models become more capable, steering techniques could play an important role in inspecting, auditing, and intervening in how models reason and act. We see this approach—using just one tool from the mechanistic interpretability toolbox—as a step toward a broader class of transparent, steerable, and semantically grounded control tools for robotics.

6 Limitations

This work introduces one of the first interpretable control knobs for foundation models in robotics, enabling behavior to be modulated through semantically grounded activations without retraining. Establishing such control-layer transparency is a step toward safe, adaptive embodied AI. While the results are promising, several challenges remain, including reliably mapping internal structure to behavior and scaling to more general settings. Addressing these challenges presents concrete opportunities to advance the robustness, generality, and practical utility of activation-based steering.

Semantic Ambiguity and Representational Drift. Our clustering method is based on token-level semantic similarity rather than direct behavioral outcomes. For example, approaches like kNN over token-projected value vectors assume that local similarity implies shared semantics. In practice, clusters can conflate distinct behaviors—e.g., "slow and careful" versus "slow and stuck"—leading to inconsistent effects. Furthermore, we found that the meaning of value vectors can shift across models, tasks, and time. A vector aligned with "up" in one context may behave differently in another. Understanding and mitigating this representational drift will be critical for developing steering interventions that transfer reliably across domains and deployment scenarios.

Fine-Tuning and Steerability. We do not yet fully understand how VLA fine-tuning affects steerability. While some semantic directions appear to retain causal influence after adaptation, it's possible that fine-tuning alters internal representations in ways that make VLM-pretrained concepts harder to access or less aligned with behavior. Investigating how steerable directions evolve with continued training is an ideal direction for future work.

Evaluation Scope and Generalization. Our evaluations, while spanning both simulation and hardware, are limited to pick-and-place tasks with a robot arm. Extending to mobile and bimanual platforms, as well as unstructured environments, will be important for validating control stability, generalization under physical variation, and alignment with human intent across real-world settings.

Acknowledgments

This work is supported by the NSF Safe Learning Enabled Systems Program, the DARPA Assured Neuro Symbolic Learning and Reasoning Program, and the ONR project "Leveraging Egocentric and Allocentric Representations for Navigation (LEARN)".

Bear Häon was additionally supported by the Schmidt Futures Quad Fellowship, NSF DToD Fellowship, and Foresight Institute Fellowship during the development of this work. He thanks the ERA Fellowship at the University of Cambridge for early exposure to mechanistic interpretability during his time as a 2023 AI Safety Fellow.

References

[1] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, L. X. Shi, J. Tanner, Q. Vuong, A. Walling, H. Wang, and U. Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164.

[2] K. Pertsch, K. Stachowicz, B. Ichter, D. Driess, S. Nair, Q. Vuong, O. Mees, C. Finn, and S. Levine. FAST: Efficient action tokenization for vision-language-action models, 2025. URL https://arxiv.org/abs/2501.09747.

[3] G. R. Team, S. Abeyruwan, J. Ainslie, J.-B. Alayrac, M. G. Arenas, T. Armstrong, A. Balakrishna, R. Baruch, M. Bauza, M. Blokzijl, S. Bohez, K. Bousmalis, A. Brohan, T. Buschmann, A. Byravan, S. Cabi, K. Caluwaerts, F. Casarini, O. Chang, J. E. Chen, X. Chen, H.-T. L. Chiang, K. Choromanski, D. D'Ambrosio, S. Dasari, T. Davchev, C. Devin, N. D. Palo, T. Ding, A. Dostmohamed, D. Driess, Y. Du, D. Dwibedi, M. Elabd, C. Fantacci, C. Fong, E. Frey, C. Fu, M. Giustina, K. Gopalakrishnan, L. Graesser, L. Hasenclever, N. Heess, B. Hernaez, A. Herzog, R. A. Hofer, J. Humplik, A. Iscen, M. G. Jacob, D. Jain, R. Julian, D. Kalashnikov, M. E. Karagozler, S. Karp, C. Kew, J. Kirkland, S. Kirmani, Y. Kuang, T. Lampe, A. Laurens, I. Leal, A. X. Lee, T.-W. E. Lee, J. Liang, Y. Lin, S. Maddineni, A. Majumdar, A. H. Michaely, R. Moreno, M. Neunert, F. Nori, C. Parada, E. Parisotto, P. Pastor, A. Pooley, K. Rao, K. Reymann, D. Sadigh, S. Saliceti, P. Sanketi, P. Sermanet, D. Shah, M. Sharma, K. Shea, C. Shu, V. Sindhwani, S. Singh, R. Soricut, J. T. Springenberg, R. Sterneck, R. Surdulescu, J. Tan, J. Tompson, V. Vanhoucke, J. Varley, G. Vesom, G. Vezzani, O. Vinyals, A. Wahid, S. Welker, P. Wohlhart, F. Xia, T. Xiao, A. Xie, J. Xie, P. Xu, S. Xu, Y. Xu, Z. Xu, Y. Yang, R. Yao, S. Yaroshenko, W. Yu, W. Yuan, J. Zhang, T. Zhang, A. Zhou, and Y. Zhou. Gemini robotics: Bringing ai into the physical world, 2025. URL https://arxiv.org/abs/2503.20020.

[4] C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, S. Johnston, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

[5] T. Bricken, A. Templeton, J. Batson, B. Chen, A. Jermyn, T. Conerly, N. Turner, C. Anil, C. Denison, A. Askell, R. Lasenby, Y. Wu, S. Kravec, N. Schiefer, T. Maxwell, N. Joseph, Z. Hatfield-Dodds, A. Tamkin, K. Nguyen, B. McLean, J. E. Burke, T. Hume, S. Carter, T. Henighan, and C. Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

[6] H. Cunningham, A. Ewart, L. Riggs, R. Huben, and L. Sharkey. Sparse autoencoders find highly interpretable features in language models, 2023. URL https://arxiv.org/abs/2309.08600.

[7] S. Marks, J. Treutlein, T. Bricken, J. Lindsey, J. Marcus, S. Mishra-Sharma, D. Ziegler, E. Ameisen, J. Batson, T. Belonax, S. R. Bowman, S. Carter, B. Chen, H. Cunningham, C. Denison, F. Dietz, S. Golechha, A. Khan, J. Kirchner, J. Leike, A. Meek, K. Nishimura-Gasparian, E. Ong, C. Olah, A. Pearce, F. Roger, J. Salle, A. Shih, M. Tong, D. Thomas, K. Rivoire, A. Jermyn, M. MacDiarmid, T. Henighan, and E. Hubinger. Auditing language models for hidden objectives, 2025. URL https://arxiv.org/abs/2503.10965.

[8] M. Geva, A. Caciularu, K. R. Wang, and Y. Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space, 2022. URL https://arxiv.org/abs/2203.14680.

[9] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. doi:10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

[10] J. Lindsey, W. Gurnee, E. Ameisen, B. Chen, A. Pearce, N. L. Turner, C. Citro, D. Abrahams, S. Carter, B. Hosmer, J. Marcus, M. Sklar, A. Templeton, T. Bricken, C. McDougall, H. Cunningham, T. Henighan, A. Jermyn, A. Jones, A. Persic, Z. Qi, T. B. Thompson, S. Zimmerman, K. Rivoire, T. Conerly, C. Olah, and J. Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

[11] K. Meng, D. Bau, A. Andonian, and Y. Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.

[12] T.-H. Wang, W. Xiao, T. Seyde, R. Hasani, and D. Rus. Measuring interpretability of neural policies of robots with disentangled representation. In *Conference on Robot Learning*, pages 602–641. PMLR, 2023.

[13] S. Pohland and C. Tomlin. Understanding the dependence of perception model competency on regions in an image. In *World Conference on Explainable Artificial Intelligence*, pages 130–154. Springer, 2024.

[14] C. Glanois, P. Weng, M. Zimmer, D. Li, T. Yang, J. Hao, and W. Liu. A survey on interpretable reinforcement learning. *Machine Learning*, 113(8):5847–5890, 2024.

[15] K. Park, Y. J. Choe, and V. Veitch. The linear representation hypothesis and the geometry of large language models. In *International Conference on Machine Learning*, pages 39643–39666. PMLR, 2024.

[16] M. Geva, R. Schuster, J. Berant, and O. Levy. Transformer feed-forward layers are key-value memories, 2021. URL https://arxiv.org/abs/2012.14913.

[17] E. Collaboration, A. O'Neill, A. Rehman, A. Gupta, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, A. Tung, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Gupta, A. Wang, A. Kolobov, A. Singh, A. Garg, A. Kembhavi, A. Xie, A. Brohan, A. Raffin, A. Sharma, A. Yavary, A. Jain, A. Balakrishna, A. Wahid, B. Burgess-Limerick, B. Kim, B. Schölkopf, B. Wulfe, B. Ichter, C. Lu, C. Xu, C. Le, C. Finn, C. Wang, C. Xu, C. Chi, C. Huang, C. Chan, C. Agia, C. Pan, C. Fu, C. Devin, D. Xu, D. Morton, D. Driess, D. Chen, D. Pathak, D. Shah, D. Büchler, D. Jayaraman, D. Kalashnikov, D. Sadigh, E. Johns, E. Foster, F. Liu, F. Ceola, F. Xia, F. Zhao, F. V. Frujeri, F. Stulp, G. Zhou, G. S. Sukhatme, G. Salhotra, G. Yan, G. Feng, G. Schiavi, G. Berseth, G. Kahn, G. Yang, G. Wang, H. Su, H.-S. Fang, H. Shi, H. Bao, H. B. Amor, H. I. Christensen, H. Furuta, H. Bharadhwaj, H. Walke, H. Fang, H. Ha, I. Mordatch, I. Radosavovic, I. Leal, J. Liang, J. Abou-Chakra, J. Kim, J. Drake, J. Peters, J. Schneider, J. Hsu, J. Vakil, J. Bohg, J. Bingham,

J. Wu, J. Gao, J. Hu, J. Wu, J. Wu, J. Sun, J. Luo, J. Gu, J. Tan, J. Oh, J. Wu, J. Lu, J. Yang, J. Malik, J. Silvério, J. Hejna, J. Booher, J. Tompson, J. Yang, J. Salvador, J. J. Lim, J. Han, K. Wang, K. Rao, K. Pertsch, K. Hausman, K. Go, K. Gopalakrishnan, K. Goldberg, K. Byrne, K. Oslund, K. Kawaharazuka, K. Black, K. Lin, K. Zhang, K. Ehsani, K. Lekkala, K. Ellis, K. Rana, K. Srinivasan, K. Fang, K. P. Singh, K.-H. Zeng, K. Hatch, K. Hsu, L. Itti, L. Y. Chen, L. Pinto, L. Fei-Fei, L. Tan, L. J. Fan, L. Ott, L. Lee, L. Weihs, M. Chen, M. Lepert, M. Memmel, M. Tomizuka, M. Itkina, M. G. Castro, M. Spero, M. Du, M. Ahn, M. C. Yip, M. Zhang, M. Ding, M. Heo, M. K. Srirama, M. Sharma, M. J. Kim, N. Kanazawa, N. Hansen, N. Heess, N. J. Joshi, N. Suenderhauf, N. Liu, N. D. Palo, N. M. M. Shafiullah, O. Mees, O. Kroemer, O. Bastani, P. R. Sanketi, P. T. Miller, P. Yin, P. Wohlhart, P. Xu, P. D. Fagan, P. Mitrano, P. Sermanet, P. Abbeel, P. Sundaresan, Q. Chen, Q. Vuong, R. Rafailov, R. Tian, R. Doshi, R. Mart'in-Mart'in, R. Baijal, R. Scalise, R. Hendrix, R. Lin, R. Qian, R. Zhang, R. Mendonca, R. Shah, R. Hoque, R. Julian, S. Bustamante, S. Kirmani, S. Levine, S. Lin, S. Moore, S. Bahl, S. Dass, S. Sonawani, S. Tulsiani, S. Song, S. Xu, S. Haldar, S. Karamcheti, S. Adebola, S. Guist, S. Nasiriany, S. Schaal, S. Welker, S. Tian, S. Ramamoorthy, S. Dasari, S. Belkhale, S. Park, S. Nair, S. Mirchandani, T. Osa, T. Gupta, T. Harada, T. Matsushima, T. Xiao, T. Kollar, T. Yu, T. Ding, T. Davchev, T. Z. Zhao, T. Armstrong, T. Darrell, T. Chung, V. Jain, V. Kumar, V. Vanhoucke, W. Zhan, W. Zhou, W. Burgard, X. Chen, X. Chen, X. Wang, X. Zhu, X. Geng, X. Liu, X. Liangwei, X. Li, Y. Pang, Y. Lu, Y. J. Ma, Y. Kim, Y. Chebotar, Y. Zhou, Y. Zhu, Y. Wu, Y. Xu, Y. Wang, Y. Bisk, Y. Dou, Y. Cho, Y. Lee, Y. Cui, Y. Cao, Y.-H. Wu, Y. Tang, Y. Zhu, Y. Zhang, Y. Jiang, Y. Li, Y. Li, Y. Iwasawa, Y. Matsuo, Z. Ma, Z. Xu, Z. J. Cui, Z. Zhang, Z. Fu, and Z. Lin. Open x-embodiment: Robotic learning datasets and rt-x models, 2024. URL https://arxiv.org/abs/2310.08864.

[18] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, P. Florence, C. Fu, M. G. Arenas, K. Gopalakrishnan, K. Han, K. Hausman, A. Herzog, J. Hsu, B. Ichter, A. Irpan, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, L. Lee, T.-W. E. Lee, S. Levine, Y. Lu, H. Michalewski, I. Mordatch, K. Pertsch, K. Rao, K. Reymann, M. Ryoo, G. Salazar, P. Sanketi, P. Sermanet, J. Singh, A. Singh, R. Soricut, H. Tran, V. Vanhoucke, Q. Vuong, A. Wahid, S. Welker, P. Wohlhart, J. Wu, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich. RT-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL https://arxiv.org/abs/2307.15818.

[19] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. P. Foster, P. R. Sanketi, Q. Vuong, et al. Openvla: An open-source vision-language-action model. In *8th Annual Conference on Robot Learning*, 2024.

[20] L. Beyer, A. Steiner, A. S. Pinto, A. Kolesnikov, X. Wang, D. Salz, M. Neumann, I. Alabdulmohsin, M. Tschannen, E. Bugliarello, T. Unterthiner, D. Keysers, S. Koppula, F. Liu, A. Grycner, A. Gritsenko, N. Houlsby, M. Kumar, K. Rong, J. Eisenschlos, R. Kabra, M. Bauer, M. Bošnjak, X. Chen, M. Minderer, P. Voigtlaender, I. Bica, I. Balazevic, J. Puigcerver, P. Papalampidi, O. Henaff, X. Xiong, R. Soricut, J. Harmsen, and X. Zhai. PaliGemma: A versatile 3b vlm for transfer, 2024. URL https://arxiv.org/abs/2407.07726.

[21] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

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

[23] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023. URL https://arxiv.org/abs/2306.03310.

[24] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

[25] N. Shazeer. GLU variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202.

A Steering Intervention Details

Let $x \in \mathbb{R}^n$ be the residual input to a transformer FFN. As shown in Equation (2), the FFN output is a sum of fixed value vectors $w_\theta^{(i)} \in \mathbb{R}^n$ weighted by input-dependent activations $f_\theta(x) \in \mathbb{R}^m$. Motivated by evidence that individual FFN neurons encode semantic features [16, 5], we override a subset $\mathcal{S} \subseteq \{1, \ldots, m\}$ of activations using a fixed scalar $\alpha \in \mathbb{R}$. Each $\mathcal{S}$ corresponds to an interpretable neuron cluster aligned with a control concept - e.g., *fast*, *up*, *careful* - identified by grouping neurons with similar token projections (Figure 1, steps 1–3), either manually or via kNN over semantic embeddings.

$$\tilde{f}_\theta^{(i)}(x) = \begin{cases} \alpha & \text{if } i \in \mathcal{S} \\ [f_\theta(x)]_i & \text{otherwise} \end{cases} \tag{5}$$

$$\text{FFN}_{\text{steered}}(x) = \sum_{i=1}^{m} \tilde{f}_\theta^{(i)}(x) \cdot w_\theta^{(i)} \tag{6}$$

This induces a residual shift $\Delta x = \text{FFN}_{\text{steered}}(x) - \text{FFN}(x)$ that propagates through the transformer and modulates the final VLA action token distribution.

Implementation. In both π_0 and OpenVLA, the FFN layers use a GEGLU activation function [25] which has the form $[f_\theta(x)]_i = [\text{GELU}(W_1 x)]_i \cdot [W_2 x]_i$, where $W_1, W_2 \in \mathbb{R}^{mxn}$ are additional parameter matrices. To implement the steering intervention, for π_0 (JAX), the neuron indices $\mathcal{S}$ and activation coefficient α are passed into modified FFN code that implements Equation (5). For OpenVLA (PyTorch), we apply a forward hook on the FFN's `down_proj` to overwrite activations.

We also experimented with a modified intervention to OpenVLA which instead applies the forward hook to the FFN's `gate_proj`. This is equivalent to the following:

$$\tilde{f}_\theta^{(i)}(x) = \begin{cases} \text{GELU}(\alpha) \cdot [W_2 x]_i & \text{if } i \in \mathcal{S} \\ [f_\theta(x)]_i & \text{otherwise} \end{cases} \tag{7}$$

We found that this intervention performed similarly to the intervention defined in Equation (5) during our exploratory experiments.

B Interpretability Experiments

B.1 Patterns in Value Vectors

Following [8], we classified a value vector as meaningful if 4 of its top 30 predicted tokens followed a common pattern. Furthermore, we classified patterns as "non-semantic" if they could be explained by a shallow syntax-level rule, such as words starting with the same 3 letters, and "semantic" otherwise. If multiple different patterns were found, we selected the one containing the highest-ranked token for determining the classification of the vector. Some examples of vectors from each category are show in table 1a (PaliGemma VLM) and 1b (π_0 VLA). Both models have 18 layers.

B.2 DROID Fine-tuning

To understand whether the contents of the task instructions in the DROID training dataset affected the value vectors of the model checkpoint fine-tuned on DROID data, we examined whether common instruction tokens became more common in the value vectors after fine-tuning. Specifically, we first tokenized all task instructions (including multiple alternative instructions for the same demonstration where available) using the PaliGemma tokenizer. We then took the top 200 most frequent instruction tokens and examined the number of times they appeared in the top-100 tokens for each value vector in the π_0-FAST and π_0-FAST-DROID-finetune model checkpoints. We assign a z-score to each token based on a two-proportion z-test and plot the results in 8. A positive z-score indicates the token is more common in the value vectors of π_0-FAST-DROID-finetune compared to π_0-FAST,

Layer	Pattern type	Pattern description	Top 10 Tokens
3	None	N/A	strick, campa, laun, increa, accla, reluct, embra, exem, desir, fath
6	Non-semantic	Camel case variable names	<bos>, EndGlobalSection, Clik, expandindo, SourceChecksum, NSCoder, EditorBrowsable, UnusedPrivate, nahilalakip, ScopeManager
6	Semantic	Image-sourcing websites	pixabay, gettyimages, shutterstock, unsplash, uefa, Ferdin, plak, makro, Hæ, Lombar
14	Semantic	Mice and keyboards	mouse, keyboard, keys, KEY, keyboard, Mouse, keys, Mouse, mouse, clavier

(a) Example value vectors and their top tokens from the PaliGemma VLM.

Layer	Pattern type	Pattern description	Top 10 Tokens
1	Non-semantic	Action tokens	<Ac0701>, <Ac0689>, <Ac0706>, <Ac0695>, <Ac0714>, <Ac0719>, <Ac0684>, <Ac0688>, <Ac0305>, <Ac0334>
14	None	N/A	maneu, shenan, affor, impra, increa, encomp, fortn, accla, philanth, volunte
15	Semantic	Strong	tough, strength,tough, toughness, toughest, tougher, strong,strength, strongest,strong
18	Semantic	Given names	javier, jorge, alberto, felipe, Áng, ricardo, eduardo, roberto, sergio, fernando

(b) Example value vectors and their top tokens from the π_0 VLA.

while a negative score indicates that it is more rare in π_0-FAST-DROID-finetune. We find that the majority of the top instruction tokens are indeed more common in the π_0-FAST-DROID-finetune value vectors, with a mean z-score of 1.0 and a mean count ratio of 1.2. While some tokens do have negative z-scores, we hypothesize that these tokens may be even more common in the base π_0 training data instructions, which unfortunately we do not have access to. Overall, these results suggest that semantic instruction inputs during VLA training may contribute to these tokens being retained in the value vectors, despite the VLA model being trained to only output action tokens.

Simulation Experiments

B.3 KNN for Temporal Depth Interventions

1. Semantic Embedding Construction. To assign meaning to each value vector, we project it into the model's output token space using the language modeling head, as described in Section 3. For each value vector $w_\theta^{(i)}$, we compute token logits and identify the top-5 tokens it most strongly activates. We then construct a semantic embedding by computing the softmax-weighted average of those token embeddings:

$$\mathbf{e}_{\text{sem}}^{(i)} = \sum_{j \in t_i} \text{softmax}(l_j^{(i)}) \cdot W_j$$

where t_i are the top token indices for value vector $w_\theta^{(i)}$, $l_j^{(i)}$ is the logit that $w_\theta^{(i)}$ assigns to token j, and W_j is the output embedding vector corresponding to token j. This yields a normalized, interpretable embedding for each value vector.

2. Depth Partitioning. The full OpenVLA model contains 352,255 value vectors across all transformer layers. To analyze temporal structure, we partition them: the first half (176,128 vectors) represents *early-layer* vectors, while the second half represents *late-layer* vectors.

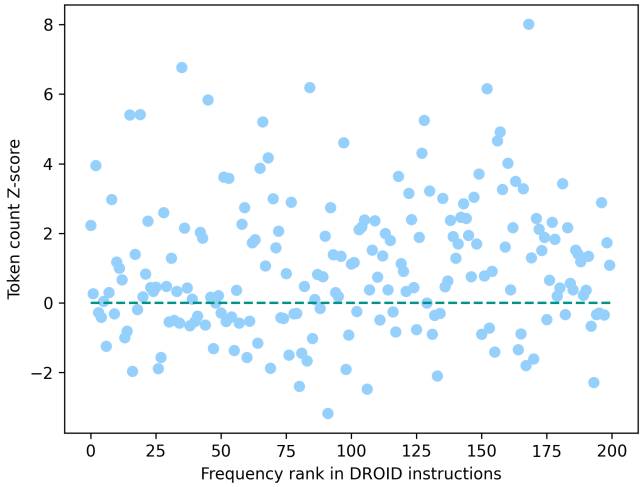


Figure 8: **Common DROID instruction tokens become more common in the value vectors of a model fine-tuned on the DROID dataset.** The majority of the top-200 instruction tokens are more common in the value vectors of the π_0-FAST-DROID-finetune checkpoint, corresponding to a positive z-score (lying above the teal dashed line).

3. kNN Clustering. We apply cosine-based k-nearest neighbor (kNN) clustering to these semantic embeddings using the cuML GPU-accelerated library. Each value vector is assigned a cluster based on its k nearest neighbors ($k \in \{10, 20, 40\}$). For each cluster, we compute a semantic centroid by averaging its members' embeddings.

4. Concept-Aligned Cluster Selection. To find clusters aligned with a target concept (e.g., "up"), we tokenize the concept word/phrase using the model's tokenizer, embed it using the language modeling head, and compare it to all cluster centroids using cosine similarity. The cluster with the highest similarity is selected for activation.

5. Temporal Partitioning for Layer-Specific Interventions. To isolate temporal effects, the above clustering pipeline was run independently on the *early-layer* and *late-layer* vector subsets. This ensured that the selected clusters were sourced exclusively from the intended depth region. For full-model interventions, clustering was performed on the full set of value vectors.

Vector	Top 5 Tokens
153784	up, [action: 0.428], Up, oks, cker
263055	up, Up, Up, up, pto
276423	up, coming, Up, up, forth
225616	up, left, Up, up, Up
274043	up, Up, Up, up, UP
285310	up, Up, Up, up, UP
287766	up, Up, Up, up, UP
287383	up, Up, Up, up, UP
263704	Up, up, Up, up, Down
323343	up, Up, Up, up, UP

Figure 9: Top-5 token projections for the most semantically aligned **"up"** clusters from the full model using kNN clustering with $k = 10$.

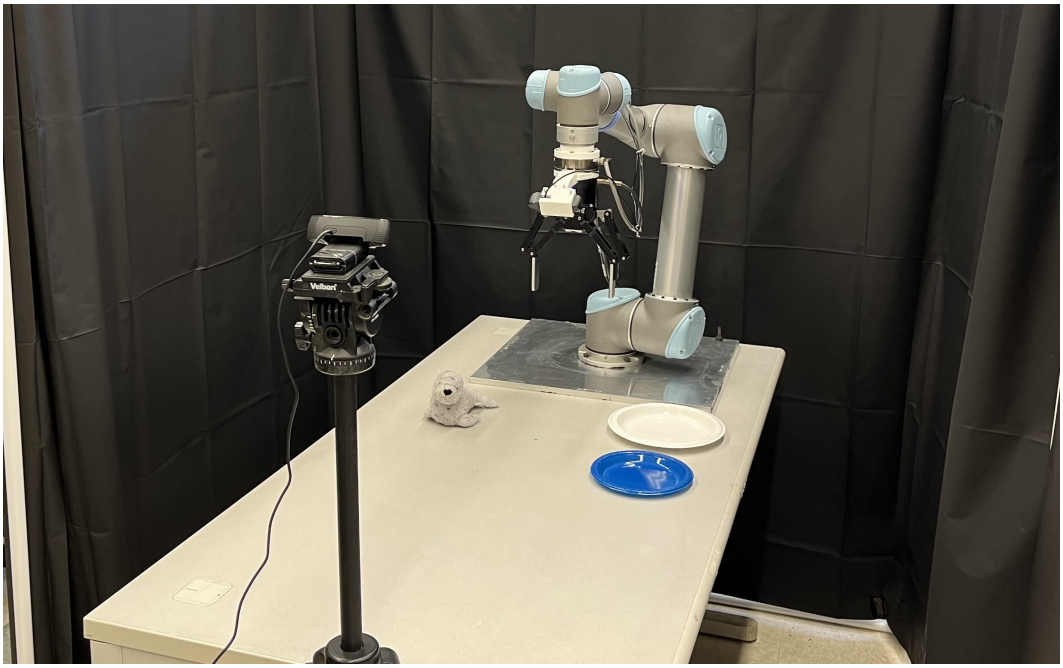
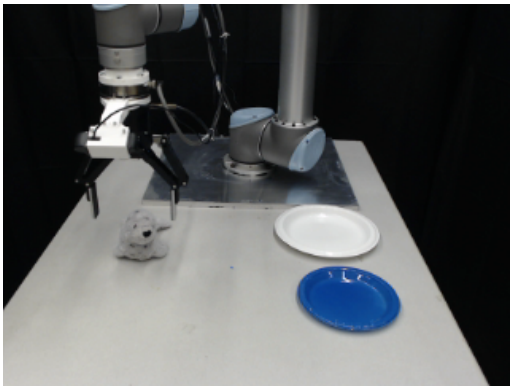
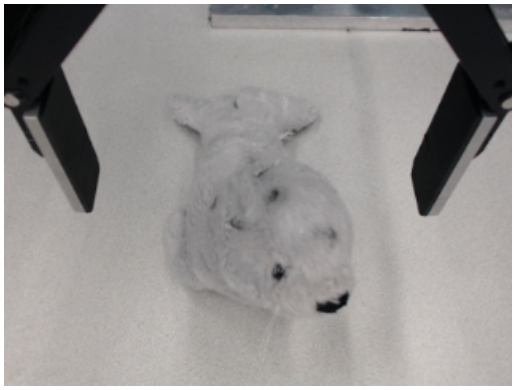


Figure 10: **Robot Setup:** Our hardware experiments use a UR5 robot arm equipped with a Robotiq 2F-140 gripper. The setup includes two cameras: a static scene camera overlooking the workspace and a wrist-mounted camera facing the gripper.



(a) Scene Camera



(b) Wrist Camera

Figure 11: Example of raw images that are input into π_0-FAST.

C Hardware Experiments

C.1 Data Collection Details

During data collection and evaluation, objects for both tasks are placed within a consistent area with slight variations in position. In both tasks, the robot begins from the same initial pose with a small random deviation in position. During data collection, the end-effector is controlled in Cartesian space using a joystick. Data is recorded at 10 Hz, including images from the wrist-mounted and scene cameras, arm joint positions, gripper position, and the corresponding target joint and gripper commands at each timestep. Example images of both cameras are shown in Figure 11.

For *Low/High Transport*, we collect 75 episodes in total—25 each where the robot transports the toy penguin at a low, medium, or high height. Each episode is labeled with the prompt, "place penguin

in basket". For *Slow/Fast Transport*, we collect 20 episodes each with a slow, medium, or fast speed and placing the seal on either a blue or white plate, totaling 120 episodes. Each episode is labeled either with the prompt "place seal on blue plate" or "place seal on white plate" depending on which plate color the seal is placed on.

C.2 Finetuning π_0-FAST

We use LoRA for fine-tuning π_0-FAST and adopt the default training configuration provided by the official π_0-FAST codebase[1]. Fine-tuning is performed separately for each task, resulting in two distinct models—one for the *Low/High Transport* task and one for the *Slow/Fast Transport* task. For both tasks, the model input consists of the UR5's six joint positions, the gripper position, and both scene and wrist camera images. The model outputs an action chunk as a sequence of joint and gripper positions. We fine-tune each model for 5000 steps using a batch size of 32 and an action chunk size of 10. We use preset normalization values provided by π_0-FAST specifically for the UR5e robot arm.

C.3 Steering Intervention and Baselines

Steering Intervention: We hand-select and upweight a cluster of semantically meaningful value vectors by a coefficient of 10. Since each value vector can be interpreted as a probability distribution over output tokens, we examine the top-10 tokens with the highest probabilities for each vector. We then identify the top six value vectors containing the highest frequency of task-relevant keywords. For *Low/High Transport* we search for the keyword "low" for the low intervention and "high" for the high intervention. For *Slow/Fast Transport* we search for keywords "slow" and "safe" for the slow intervention and "fast" and "risk" for the fast intervention. We provide the value vectors used in the hardware experiments and their top-10 tokens in Table 2.

Random Intervention Baseline: We randomly select a cluster of six value vectors and upweight them by a coefficient of 10. We keep these vectors the same for the whole evaluation.

No Intervention Baseline: We run inference on π_0-FAST without any modification and use the same prompt as during fine-tuning: "place penguin in basket" for *Low/High Transport* and "place seal on blue plate" for *Slow/Fast Transport*.

Prompt Modification Baseline: We prepend steering-related keywords to the original prompts used during finetuning. For *Low/High Transport*, the prompt "place penguin in basket" is modified to "low place penguin in basket" or "high place penguin in basket" for the low and high interventions, respectively. For *Slow/Fast Transport*, the prompt "place seal on blue plate" is modified to "slow safe place seal on blue plate" or "fast risk place seal on blue plate" for the slow and fast interventions, respectively.

[1] https://github.com/Physical-Intelligence/openpi

Vector	Top 10 Tokens
5674	low, low, Low, Low, lower, LOW, lower, lowest, LOW, lows
253282	low, low, Low, Low, LOW, LOW, Lower, lower, Lower, lowest
263305	low, high, low, high, High, High, LOW, Low, Low, LOW
230595	low, medium, LOW, high, low, MEDIUM, Low, Low, Medium, medium
256528	low, Low, fillType, low, Low, StoreMessageInfo, PerformLayout, LOW, lows, WithIOException
246672	low, lows, low, Darío, cushi, Valentín, ecru, LOW, LOW, chaub

(a) Low intervention for *Low/High Transport*.

Vector	Top 10 Tokens
260217	high, high, High, High, HIGH, HIGH, low, low, Low, LOW
265014	high, high, High, High, HIGH, HIGH, hight, shenan, intersper, 高
266729	high, high, High, High, HIGH, HIGH, highs, 高, middle, hight
267454	high, High, High, high, HIGH, hight, HIGH, Odys, hig, quitted
270442	high, high, High, High, HIGH, HIGH, hight, higher, inappro, unwarran
275165	High, high, high, High, HIGH, HIGH, yüksek, 高, hoog, hight

(b) High intervention for *Low/High Transport*.

Vector	Top 10 Tokens
242252	safe, safe, safely, Safe, Safe, SAFE, SAFE, safety, safer, <bos>
104785	çük, SLOW, slow, Slow, slow, Slow, maksi, lacable, jät, tempo
243496	safe, safe, SAFE, ArgumentParser, Safe, Safe, <bos>, Confira, memoized, Ainda
73159	soeur, slow, slowest, calm, Slow, slow, SLOW, calmer, hairc, atguigu
230490	safe, safety, safe, Safe, Safe, Safety, Safety, safety, safely, SAFETY
234025	slow, slowly, slow, <bos>, <Ac1403>, <Ac1327>, Slow, Slow, <Ac1340>, <Ac1450>

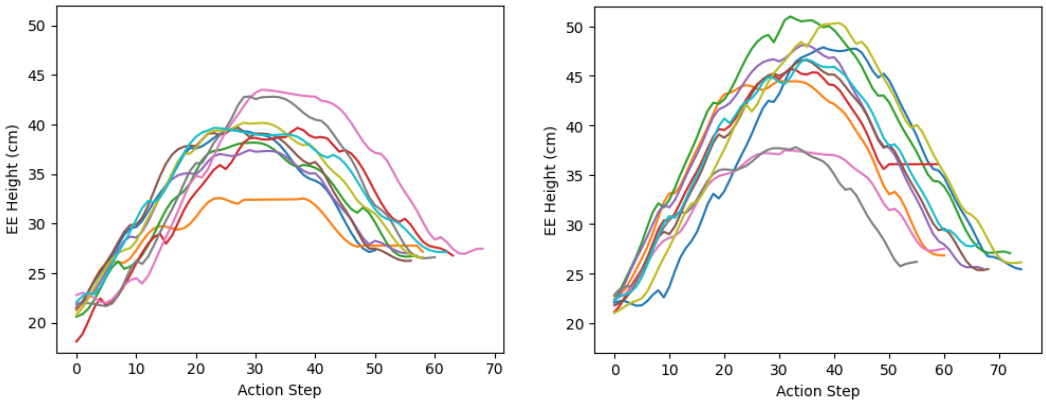
(c) Slow intervention for *Slow/Fast Transport*.

Vector	Top 10 Tokens
239787	risk, risk, <bos>, Risk, Risk, RISK, risks, Risks, RISK, riesgo
240184	fast, fast, FAST, Fast, Fast, FAST, quickly, tolerably, hentai, Quickly
259344	risk, risk, Risk, Risk, risks, Risks, riesgo, RISK, RISK, risky
262123	fast, fast, Fast, Fast, faster, FAST, fastest, quick, quicker, FAST
271027	fast, fast, fastest, faster, Fast, Fast, FAST, Faster, faster, FAST
273133	fast, fast, Fast, Fast, FAST, FAST, <Ac1446>, faster, <Ac0390>, veloce

(d) Fast intervention for *Slow/Fast Transport*.

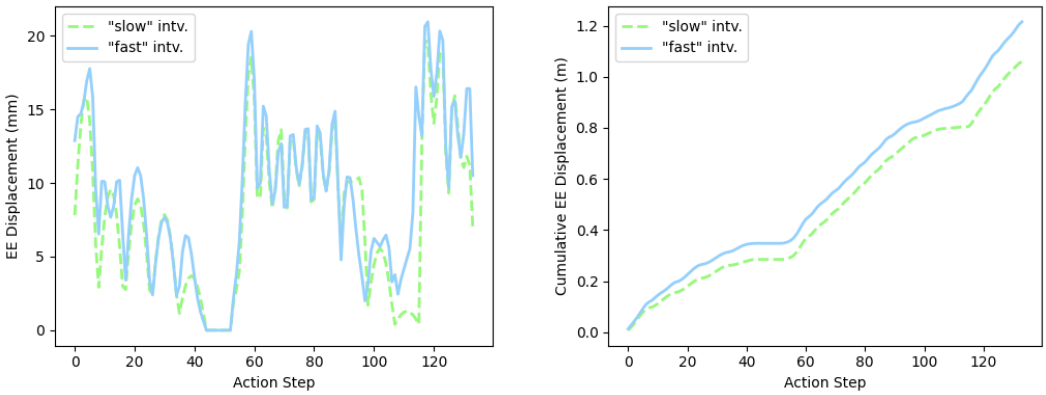
Table 2: Value vectors and their top-10 highest probability output tokens for steering interventions in *Low/High Transport* and *Slow/Fast Transport*.

C.4 Steering Intervention Trajectories



(a) Low Intervention (b) High Intervention

Figure 12: **Low/High Transport:** End-effector height from 10 trajectories with low intervention (a) vs. high intervention (b). We plot the segment of the trajectory from when the robot picks up the penguin to when it releases it into the basket. Low interventions have a lower maximum height on average.



(a) End-effector Displacement (b) Cumulative End-effector Displacement

Figure 13: **Slow/Fast Transport:** End-effector displacement (a) and cumulative end-effector displacement (b) over a single trajectory with slow intervention vs. high intervention. While it may be hard to compare displacement at each action step, the cumulative displacement shows how the slow intervention reduces displacement.