CAUSAL ANALYSIS OF ROBOT FOUNDATION MODELS ENABLES EFFICIENCY GAINS

Anonymous authors
Paper under double-blind review

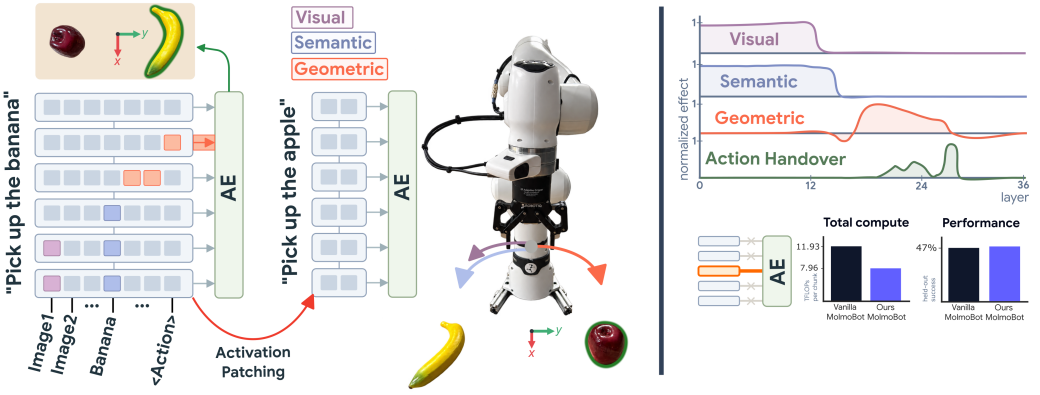


Figure 1: **Activation patching reveals how a vision-language-action (VLA) model turns perception into action.** (**Left**) We copy activations from a donor episode ("Pick up the banana") into a target episode ("Pick up the apple") with the position of the objects swapped. Visual information (purple) sits in early image tokens, semantic information (blue) in the early noun token, and geometric information (orange) in later text tokens. The action expert (AE) reads from a narrow band of later layer. (**Top right**) Schematic of these effects by layer. The action expert reads from a narrow band of layers.(**Bottom right**) Keeping only this interface in MolmoBot, **3 of 36 connections**, cuts compute by 33% with comparable average success in simulation (46.7% vs. 46.9%).

ABSTRACT

Vision-language-action (VLA) models are a promising approach to general purpose robot manipulation. While the internal computation of vision-language models, which compose the backbone of VLAs, has been studied extensively, less is known about how their perceptual and semantic representations are transformed into action in VLA models. We study this question mechanistically in three frontier VLA models ($\pi_{0.5}$, MolmoAct2, and MolmoBot) by analyzing the causal pathways through which the input image and the instruction shape actions. Across the three models, activation patching shows that causally relevant information influences the action expert (AE) through a narrow band of mid-to-late cross-attention layers, which we term the *action-conditioning interface*. The content of this information changes with token and layer within the model. Early-to-mid layer representations of the target noun token determine which object is selected, whereas mid-to-late layers carry the spatial information in text tokens that redirects the movement when patched. Where spatial information is stored in prompt tokens various based on the architecture. In MolmoAct2 the model forms task-conditioned representations at the text positions that follow the instruction, whereas $\pi_{0.5}$ additionally stores this information at image token positions. We show that these findings can be translated directly into improving model efficiency. We find that restricting MolmoBot's action expert conditioning to 3/36 layers reduces inference FLOPs by up to 33% while maintaining comparable task success with the default model across more than 3,000 evaluation episodes (46.7% vs. 46.9%), and the same principle carries over to real-robot deployment of $\pi_{0.5}$.

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.

1 INTRODUCTION

Vision-language-action (VLA) models are a promising route to achieve general-purpose robot manipulation. By adapting large pretrained vision-language models (VLMs) to action (Driess et al., 2023; Beyer et al., 2024; Deitke et al., 2025), VLAs combine broad visual and semantic knowledge with continuous control and increasingly handle manipulation tasks across objects, environments, and robot embodiments (Brohan et al., 2022; 2023; Kim et al., 2024; Black et al., 2025; Physical Intelligence et al., 2025). Architecturally, VLAs now typically combine a pretrained VLM backbone (Chi et al., 2023; Black et al., 2025; Physical Intelligence et al., 2025) with a separate diffusion or flow-matching action expert that produces continuous control. The two modules are joined through a dense, layer-wise interface that conditions the expert on the backbone at every layer. Despite their growing real-world efficacy and partially due to the multi-modal architecture design, these models remain black boxes. VLAs transform thousands of image, language, and proprioceptive tokens through billions of parameters before producing a small set of continuous action tokens, leaving the internal mapping from perception to action an open area of study.

One promising approach to understanding VLA computation is mechanistic interpretability (Grant et al., 2026; Bhardwaj et al., 2026; Molinari et al., 2025; Khan et al., 2025; Goyal, 2026; Yang & Mokalla, 2026). In particular, *activation patching* (Vig et al., 2020; Geiger et al., 2021; Zhang & Nanda, 2024; Heimersheim & Nanda, 2024) approaches aim to measure the *causal* role of an internal representation by replacing the target representation from one rollout with a donor from another rollout, while measuring the resulting change in behavior. In language models, such interventions have localized the computations behind factual recall and uncovered computations underlying numerous tasks (Meng et al., 2022; Wang et al., 2023; Geiger et al., 2025; Mueller et al., 2026), helping elucidate the causal relationships between internal model representations and behavior.

Recent work has begun to apply such tools to VLAs, identifying internal directions that steer movement, recovering sparse features tied to semantic concepts, and manipulating observable features to control behavior (Häon et al., 2025; Swann et al., 2026; Buurmeijer et al., 2026). Closest to our work, Grant et al. (2026) performed activation-injection experiments and showed that patching representations into a VLA can redirect movement toward the spatial location of a donor task. Shi et al. (2026) used attention knockout (Geva et al., 2023), which blocks attention from selected token positions (such as all image or all language tokens) at chosen layers and measures how the output changes, to characterize which modalities and layers action generation depends on. Together, these results establish that individual VLA representations can be causally manipulated. However, they leave open fundamental questions related to VLA computation that this work addresses: (1) how is task-relevant information composed across the depth of the backbone, (2) how does the action expert acquire this information, and (3) how can such mechanistic findings guide the design of future VLAs.

We take up these questions in three frontier open-source VLAs: $\pi_{0.5}$, MolmoBot, and MolmoAct2 (Physical Intelligence et al., 2025; Deshpande et al., 2026; Fang et al., 2026). Across all three models, we find that perception and action communicate through a surprisingly narrow channel (1-3 cross-attention layers). Causal influence concentrates in a middle-to-late band of VLM layers, which we term the *action-conditioning interface*. This suggests that the action expert ingests representations which already encode task-relevant spatial information, rather than depending equally on every stage of backbone computation. Tracing how those representations form reveals a progression across depth, from transferable object selection to spatially grounded action information. The two kinds of information also reside on different tokens: semantic content is carried by the noun token of the instruction, while geometric content is spread across the other prompt tokens.

We apply our findings of the narrow *action-conditioning-interface* between the AE and VLM to improve VLA efficiency. Many prior works have attempted to reduce VLA computation, by skipping or exiting backbone layers (Yue et al., 2024; Zhang et al., 2025; 2026b), pruning redundant layers or parameters (Yang et al., 2025; Nguyen et al., 2026; Sun et al., 2026; Zhang et al., 2026a), or conditioning the action head on only a subset of backbone layers (NVIDIA et al., 2025; Shukor et al., 2025). These methods decide what to remove using heuristics such as representational similarity or benchmark success. We offer a more mechanistic account of why these designs work. Our results show that the representations necessary for action are stored sparsely across tokens and layers.

In this work we show that by leveraging causal analysis we can choose exactly the relevant areas of the model to prune. Restricting conditioning to the causally important layers removes 92% of

MolmoBot's cross-attention parameters and reduces the estimated VLM–action-expert floating-point operation count by up to 33%. The simplified architecture achieves task success comparable to the vanilla policy across 3,000 evaluation episodes. The same approach carries over to preliminary trials on a real robot, where sparse and dense $\pi_{0.5}$ policies achieve the same success rate. Our contributions are as follows:

1. **Large-scale causal analysis across three frontier VLA models.** We build a counterfactual dataset with over 156 counterfactual pairs across sets of household objects. We run over 400,000 activation patching interventions across $\pi_{0.5}$, MolmoAct2, and MolmoBot specifically designed to differentiate representations within the backbone.
2. **Localization of visual, semantic and geometric information from the backbone to the action expert.** Visual information influences behavior through early image tokens, semantic information through the early target noun, and geometric information through later text and readout positions. Furthermore we identify a narrow middle-to-late region through which these representations condition action generation.
3. **Mechanistically informed architectural improvements.** We use the identified interface to design substantially sparser VLM–action-expert connections, preserving observed closed-loop performance while reducing the estimated combined operation count by up to 33%. We validate the simplified architecture through direct training and adaptation in simulation, and demonstrate initial transfer of the design principle to real-robot deployment with $\pi_{0.5}$.

2 METHODS

Our goal is to study how task-relevant information emerges within the vision-language backbone and is passed to the action expert. Our experiments combine activation patching at backbone and VLM-AE interface with controlled changes to object identity and spatial layout.

Models. We study MolmoBot, $\pi_{0.5}$, and MolmoAct2 (Deshpande et al., 2026; Physical Intelligence et al., 2025; Fang et al., 2026). Each model conditions its continuous action expert on representations from all backbone layers using different attention implementations. MolmoBot supplies layer-wise hidden states through cross-attention, MolmoAct2 supplies layer-wise keys and values, and $\pi_{0.5}$ conditions on a shared multimodal prefix. Their attention masks also differ: in MolmoBot and MolmoAct2, image positions cannot attend to subsequent instruction tokens, whereas $\pi_{0.5}$ uses bidirectional attention within the prefix.

Model notation. Let $x = (I, u, s)$ denote an input comprising image observations I, a language instruction u, and proprioceptive state s. The policy F produces an action sequence $\boldsymbol{a} \in \mathbb{R}^{H \times d_a}$,where H is the prediction horizon and d_a is the number of action components per timestep. We write

$$\boldsymbol{a} = F(x; \epsilon) = A\big(s, C_1(x), \ldots, C_M(x); \epsilon\big), \tag{1}$$

where A denotes the action expert together with its action-generation procedure, ϵ denotes generation noise, and $C_j(x)$ is the backbone-derived conditioning supplied to expert block $j \in \{1, \ldots, M\}$. Direct conditioning on the state s is included where used by the architecture (MolmoBot). In our experiments we hold ϵ fixed across paired runs and drop it below for simplicity.

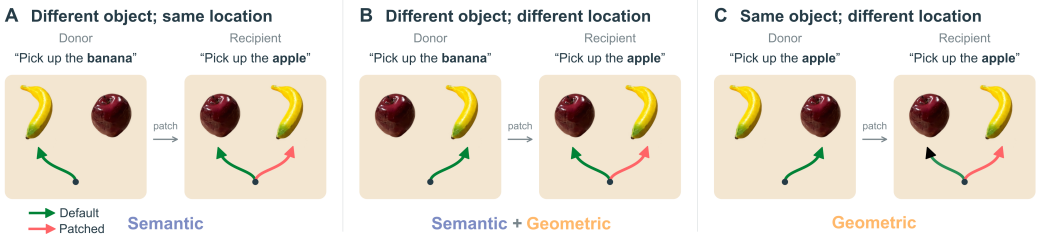
Let $z_\ell(x) \in \mathbb{R}^{N \times d_\ell}$ denote the backbone residual-stream activations at layer ℓ, with N token positions and hidden dimension d_ℓ, and let $z_{\ell,S}(x)$ denote the tokens at positions S. Here, $z_\ell(x)$ denotes residual-stream activations within backbone layer ℓ, while $C_j(x)$ denotes the conditioning supplied to action-expert block j. The conditioning $C_j(x)$ takes different forms across models. In MolmoBot, action-expert block j cross-attends to backbone hidden states; in MolmoAct2, it cross-attends to cached backbone keys and values. In $\pi_{0.5}$, cached backbone keys and values form the multimodal prefix within the expert's shared attention computation. In all cases, $C_{j,S}(x)$ denotes the conditioning restricted to context-token positions S.

2.1 ACTIVATION PATCHING.

Activation patching replaces an internal representation in a recipient run with the corresponding representation from a donor run, then measures the change in output (Vig et al., 2020; Geiger et al., 2021; 2025; Heimersheim & Nanda, 2024). Let c denote a concept of interest, such as object identity,

Comparison	Instruction	Scene layout	Requested object	Target location
A	Changes	Swapped	Different	Same
B	Changes	Fixed	Different	Different
C	Fixed	Swapped	Same	Different

(a) Donor–recipient differences across counterfactual comparisons .



(b) Predicted reaches under the labeled transfer hypotheses.

Figure 2: **Three counterfactual comparisons probe object selection and spatially specific behavior.** We patch activations from a donor run into a recipient run. (a) Changes to the instruction and scene layout yield three combinations of requested-object identity and target location. (b) Green arrows show the unpatched behavior; red arrows show predicted recipient behavior if the patch transfers the labeled information. These labels denote transfer *hypotheses*, not information uniquely isolated by single comparisons. For example, in **A**, patching scene-layout information through an image patch can produce the same behavioral shift as patching target-object identity through a noun patch. We therefore interpret effects jointly across the comparisons and patched token positions.

spatial location, or their combination, and write $c(x)$ for its value on input x. We construct recipient–donor pairs (x_r, x_d) that vary this concept, with $c(x_r) \neq c(x_d)$, while controlling other task factors. The pair defines the counterfactual contrast, and the selected layers and token positions define where we intervene. Donor and recipient token positions are aligned before patching.

Interface patching replaces the conditioning supplied to selected action-expert blocks $J \leq M$ while keeping the recipient VLM computation fixed:

$$\boldsymbol{a}^{\text{int}}_{J,S} = F(x_r)|_{C_{j,S} \leftarrow C_{j,S}(x_d),\ j \in J}\,. \tag{2}$$

Here, the intervention notation means that the recipient computation is run with the indicated internal quantities replaced by the corresponding quantities from the donor. All unpatched conditioning retains its original value, while the expert continues computing from the modified inputs. This tests where the expert reads causally relevant information from the VLM.

Backbone patching instead replaces selected residual-stream activations within the VLM:

$$\boldsymbol{a}^{\text{bb}}_{\ell,S} = F(x_r)|_{z_{\ell,S} \leftarrow z_{\ell,S}(x_d)}\,. \tag{3}$$

Other activations at the intervention site retain their original values, and all causally downstream backbone computation, interface conditioning, and action-expert computation proceed from the patched state. This tests which representations influence behavior through their downstream effects.

2.2 Counterfactual comparisons

Our goal is to distinguish representations that determine which object the model selects from those that determine where it reaches. We construct donor–recipient pairs by changing the requested object, the scene layout, or both. These changes allow us to compare the effects of transferring object-selection signals across layouts with those of transferring spatially specific information. We then vary the patched token positions to test where these forms of behavioral influence arise.

We construct two layouts containing the same two objects with their left–right arrangement exchanged, and pair each layout with instructions to pick up either object. We write the concept of interest as

$c(x) = (o(x), p(x))$, where $o(x)$ is the *identity* of the instructed object and $p(x)$ is its nominal *spatial location*. Recipient–donor pairs (x_r, x_d) yield three comparisons, illustrated in Figure 2:

1. **A: Different object, same location:** $o(x_r) \neq o(x_d)$ and $p(x_r) = p(x_d)$. Both the instruction and layout change, so the target objects occupy the same location. If the patched activations contain semantic information we expect the grasped object to redirect towards the donor object, while transfer of target location alone should preserve the recipient's target location.
2. **B: Different object, different location:** $o(x_r) \neq o(x_d)$ and $p(x_r) \neq p(x_d)$. The scene remains fixed while the instruction requests the other object. Transfer of either object identity or target location should redirect the recipient toward the donor target, so this comparison alone does not distinguish semantic from geometric information.
3. **C: Same object, different location:** $o(x_r) = o(x_d)$ and $p(x_r) \neq p(x_d)$. The instruction remains fixed while the layout changes. Transfer of target location should redirect the recipient toward the donor target's position, which is occupied by the other object in the recipient scene; transfer of object identity alone should preserve the recipient's choice of object.

For each comparison, we vary the patched token positions S, examining image tokens, the target noun, downstream prompt spans, and final readout positions. We note that individual comparisons alone do not determine what information a patch transfers. In comparison **A**, for example, a noun patch can redirect the reach by changing which object the model selects, whereas an image patch can redirect it by changing the represented location of the recipient's requested object. We therefore interpret behavioral effects jointly across counterfactual comparisons and patched token positions.

Measuring behavioral transfer. Using shared action noise ϵ across clean and patched runs, we compare the recipient output $F(x_r)$ with the patched output $\boldsymbol{a}^{\text{int}}_{J,S}$ or $\boldsymbol{a}^{\text{bb}}_{\ell,S}$. We reconstruct trajectories from predicted action commands using forward kinematics, $\boldsymbol{\tau} = \text{FK}(\boldsymbol{a})$. Let $\boldsymbol{\tau}_r$, $\boldsymbol{\tau}_{\text{patch}}$, and $\boldsymbol{\tau}_{\text{ref}}$ denote the recipient, patched, and reference trajectories, represented as vectors of stacked positions. We quantify behavioral transfer by the normalized trajectory shift

$$E = \frac{\langle \boldsymbol{\tau}_{\text{patch}} - \boldsymbol{\tau}_r,\, \boldsymbol{\tau}_{\text{ref}} - \boldsymbol{\tau}_r \rangle}{\|\boldsymbol{\tau}_{\text{ref}} - \boldsymbol{\tau}_r\|_2^2}, \tag{4}$$

defined when $\boldsymbol{\tau}_{\text{ref}} \neq \boldsymbol{\tau}_r$. Put simply, E is how far the patch moves the recipient's trajectory toward the reference, as a fraction of the full distance between them. $E = 0$ means the patch had no effect in that direction, and $E = 1$ means it moved the trajectory all the way to the reference. For interface patching, the reference is the clean donor trajectory produced by $F(x_d)$. For backbone patching, it is the clean trajectory obtained by changing the recipient instruction to request the other object while retaining the recipient scene. We also use this reference for interface patching in comparison A as the donor trajectory coincides with the recipient's.

3 Results

In a VLA, information flows from perception, through the vision-language backbone, to the action expert that produces motor commands. We now leverage the counterfactual comparisons and normalized trajectory shift score (described in Section 2) to understand how perception comes to shape action by tracing this how this information flows backward from the output. We first patch the cross attention layers connecting action expert to the backbone to find where the expert reads behaviorally relevant information (Figure 3), and then patch the backbone itself to trace how that information forms upstream (Figure 4).

3.1 Action Expert - Interface Patching

Figure 3 shows the results of counterfactual comparisons **B** and **C** across individual and three-layer-window cross attention layers between the action expert and VLM backbone (Section 3.1). We do not show comparison A, because it produces no transfer toward the other object at any single connection or three-layer window in any model ($|E| \leq 0.02$). Since A isolates semantic information, this suggests that the action expert reads positional rather than semantic information from the VLM. When the target position does change as in **B** and **C** we observe a highly uneven behavior of influence; e.g., in MolmoBot, patching the action expert at layer 24 alone recovers over 50% of the instructed

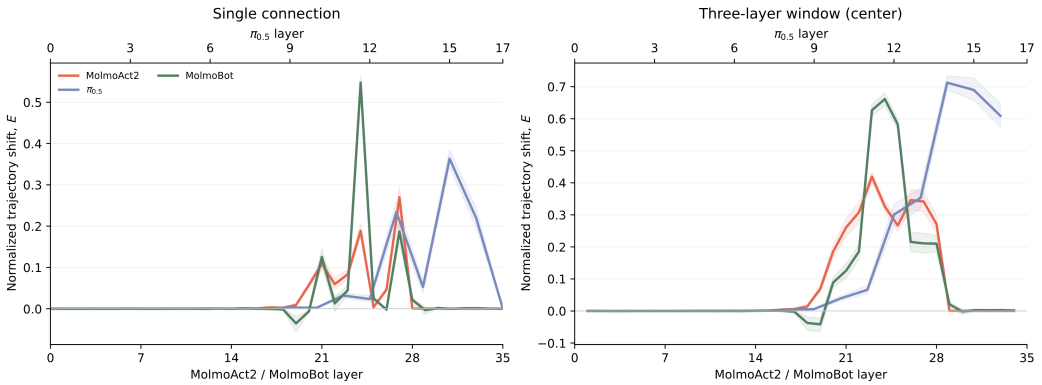


Figure 3: **Action conditioning concentrates in model-specific layer bands.** Curves show normalized trajectory shift E toward the clean donor trajectory after patching one action-expert connection (left) or three consecutive connections (right, indexed by the center layer). The results are across 144,768 patches over 26 scenes, averaging comparisons B and C.

reach contrast, and patching a three-layer window centered on layer 24 recovers close to 70%. Most other individual layers have comparatively little influence.

Importantly, we find the same pattern generalizes across the other two models: $\pi_{0.5}$ peaks at over 35% for a single-layer patch and over 70% for a sliding patch, and MolmoAct2 at nearly 30% and over 40%, respectively. In summary, in each model, a narrow band of only one to three adjacent layers accounts for most of the conditioning that shapes the action expert's behavior. We refer to this concentrated readout region as the *action-conditioning interface*.

3.2 VLM - Backbone Patching

We now turn to the VLM backbone and ask how scene information and the instruction contribute to spatially specific action conditioning. Figure 4 shows backbone-patching effects across token groups and layers for the three comparisons in Figure 2. We interpret each effect jointly from the comparison and the patched positions. Comparison **A** tests whether object selection transfers across layouts when the donor and recipient targets occupy the same location. Comparison **B** tests instruction-dependent influence in a fixed scene, but cannot alone distinguish object selection from target-location transfer. Comparison **C** tests scene-dependent influence while holding the instruction fixed. All experiments use the first timestep of each episode, when both objects are visible to both cameras.

Early image representations convey scene-layout information. When the scene layouts differ (comparisons **A** and **C**), patching early image representations strongly redirects behavior. For the dominant camera, effects approach perfect reference recovery $E = 1$ in early layers before declining at greater depth. The exterior view dominates in MolmoAct2, whereas the wrist view dominates in $\pi_{0.5}$. These effects are consistent with transferring information about the donor's scene layout. In comparison **A**, an image patch can change the represented location of the recipient's requested object, redirecting the reach while leaving the recipient's instruction unchanged.

Early noun representations support object selection across layouts. Patching early noun representations can instead change which object the model selects. Across all three models, these patches can redirect the recipient toward the donor's requested object. Comparison **A** provides the clearest evidence for this semantic transfer: the redirected reach follows that object's location in the recipient scene, even though the donor and recipient targets originally occupy the same location. For example, transferring the representation of "banana" from a scene with the banana on the left can redirect an apple-instructed recipient toward the banana on its right. The reach therefore follows the donor's requested object identity at its recipient-scene location, an effect that cannot be explained by copying the donor's target location alone.

Early noun patches also redirect behavior in comparison **B**, where the scene remains fixed and only the requested object changes. By contrast, noun patches have little effect in comparison **C** in MolmoAct2,

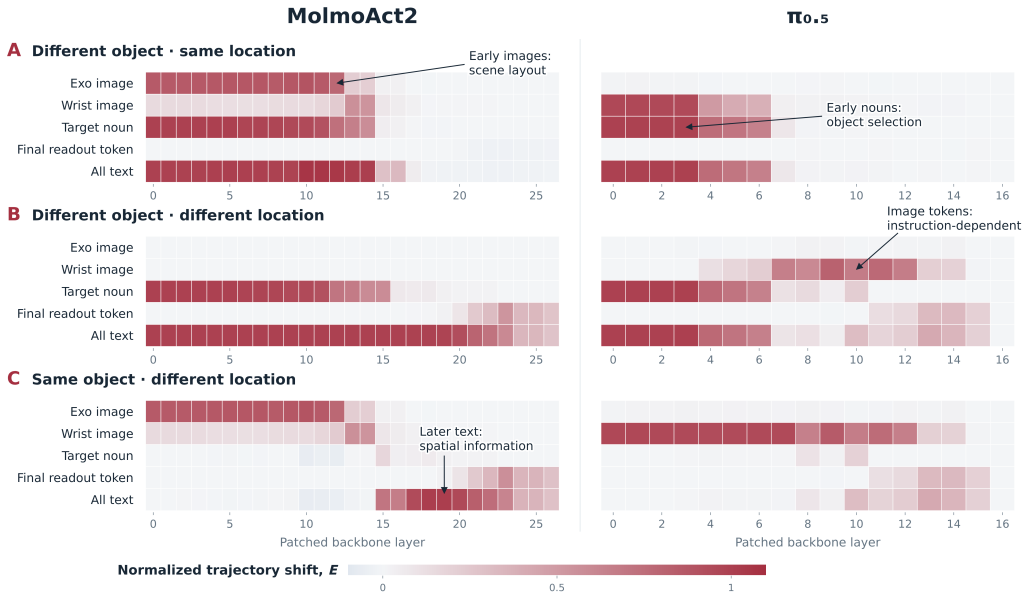


Figure 4: **Backbone patching separates object selection from scene-dependent spatial information in MolmoAct2 and $\pi_{0.5}$.** Colors show normalized trajectory shift E toward the other-object trajectory in the recipient scene. Each cell averages 416 patches: 26 donor–recipient pairs, each patched in both directions under eight noise seeds. We omit MolmoAct2 layers 27–35 as they have $|E| < 0.001$.

where the requested object remains fixed and its location changes. Together, these results support an early noun-mediated signal for object selection whose behavioral influence weakens with depth.

Later text representations support spatially specific behavior. Comparison **C** holds the instruction fixed while changing the target object's location. Patching deeper text representations redirects the reach despite identical donor and recipient text, supporting the presence of scene-dependent spatial information at these positions.

In MolmoAct2, patching all text positions produces stronger effects than patching the final readout token alone. The readout token `<action_output>` becomes an effective intervention site later, with spatial-transfer effects peaking around layers 22–23. These results indicate contributions beyond the readout token, alongside its increasing influence in deeper layers. They establish spatially specific behavioral influence, but do not distinguish an explicit coordinate representation from a spatially specific motor plan.

Instruction-dependent information also reaches image positions in $\pi_{0.5}$. Comparison **B** reveals an architectural difference in where task information becomes available. Here, donor and recipient images are identical. In MolmoAct2 and MolmoBot, image positions cannot attend to the subsequent instruction, so their activations are unchanged by the instruction swap and patching them has no effect. In $\pi_{0.5}$, patching intermediate-layer image representations redirects behavior despite the identical visual inputs. Its bidirectional prefix attention allows image positions to incorporate the instruction.

Together with the image-patching effects in comparison **C**, this result shows that $\pi_{0.5}$'s image positions support both instruction-dependent and scene-dependent behavioral influence. The relevant information is therefore not restricted to positions associated with its input modality: text positions can support spatial behavior, and image positions can become conditioned on the requested task.

Summary. Together, these experiments support a progression from early scene-layout and object-selection signals to spatially specific action conditioning in deeper layers. The information carried at a token position need not match its input modality: text and readout positions can support spatial behavior, while image positions in $\pi_{0.5}$ can become instruction-conditioned. These interventions

	Pick	Pick & Place		PnP Next-To		Avg
Cross-attention layers	Success	Oracle	Final	Oracle	Final	(oracle)
All 36 (released MolmoBot-DROID)	62.0	57.4	49.6	21.2	15.1	46.9 ± 1.7
Layers 23–25, pretrained	57.7	59.2	50.7	23.3	18.0	46.7 ± 1.7
Layer 24, pretrained	63.0	53.0	45.7	20.8	15.9	45.6 ± 1.7
Layers 23–25, adapt (50K FT)	56.8	55.4	45.9	23.1	17.8	45.1 ± 1.7
Layer 24, adapt (50K FT)	54.7	54.0	44.9	24.9	19.4	44.5 ± 1.7
Layer 12, adapt (50K FT)	25.7	20.9	16.3	6.9	4.9	17.8 ± 1.3

Table 1: **Keeping only the action-conditioning interface preserves performance** Success rate (%) on the MolmoBot paper's simulation benchmarks (1,000 identical episodes per benchmark). Reading only layer 24 (1 of 36 connections) comes within 1.3 points of the released all-layer model when trained from scratch. Reading only L23–25 comes within 0.2 points and gives the best Pick & Place success of any model. ± is a 95% binomial confidence interval. Final measures success state at end of episode. Oracle terminates at the first success state. Shading ranks each column (darkest = best).

establish transferable behavioral influence, but do not show that the relevant information is stored exclusively at the identified positions or distinguish an explicit coordinate representation from a spatially specific motor plan.

4 Efficiency Improvements in Real Policies

Our mechanistic findings suggest several opportunities to improve the efficiency of VLA architectures. Here, we focus on the clearest architectural opportunity: the narrow VLM–action-expert interface identified in Figure 3. Thus, we restrict action expert conditioning to the identified layers by removing completely the cross attention layers outside of the *action-conditioning interface*, and evaluate whether these simplified policies retain their capabilities.

4.1 MolmoBot In Simulation

We conduct our main evaluation with MolmoBot, whose openly available training data and simulation benchmarks support both training new policies and adapting an existing checkpoint (Deshpande et al., 2026). Unlike the initial-timestep patching experiments, these evaluations test closed-loop behavior throughout complete manipulation tasks. Guided by the *action-conditioning interface* located in layer-24 (Figure 3), we compare the released policy, which uses all 36 connections, against three interface variants: (i) policies trained with only the layer-24 interface, (ii) policies trained with only the layer-23-25 interface, and (iii) a policy trained with only the layer-12 in as a baseline control outside the *action-conditioning interface*. For both (i) and (ii) we compare training the model from scratch and finetuning for 50k steps from the released checkpoint.

Table 1 reports results on three simulation benchmarks, with 1,000 episodes per benchmark and identical evaluation episodes across models. These are the provided MolmoBot benchmarks and use the MolmoSpaces simulator Deshpande et al. (2026) for three tasks: Pick (pick up a target object), Pick and Place (pick up a target object and place it into a target location), and Next-To (pick up a target object and place it next to a named object). Overall, we observe that the policy trained with only the *action-conditioning interface* is comparable to the original policy, with a 33% reduction in FLOPS. The layer-24 pretrained policy achieves the highest Pick success (63.0%), the layer-23-25 policy achieves the highest Pick & Place oracle success (59.2%), while the layer-24 adapted policy achieves the highest Next-To oracle success (24.9%). Averaging Pick success and the two oracle metrics gives 46.9% for the dense policy, 46.7% for the layer-23-25 pretraining and 45.6% for layer-24 pretraining. Thus, the limited connection interface supports almost identical overall performance.

These architectures remove between 33-35 of the 36 cross-attention modules (93%-97% of the interface). Additionally, because nothing reads VLM layers 26–35, the VLM can stop early. Together, these cut the full policy's computation from 11.93 to 7.96 TFLOPs per action chunk, a 33% reduction (Section B). The savings hold on hardware. On a single H100, producing one action chunk takes 209 ms instead of 311 ms, a 33% latency reduction. At batch size 64, throughput rises from 21.5 to 30.9 chunks per second (+44%). The throughput gain matters beyond single-robot control, for

$\pi_{0.5}$ policy	Success	Median task time (s)
Original (all 18 layers)	22/25 (88%)	53.6
Restricted (layers 14–16)	22/25 (88%)	45.4

Table 2: **Real-robot evaluation of $\pi_{0.5}$.** On a fruit-in-basket task (placing three pieces of fruit into a basket), restricting action-expert attention to layers 14–16 preserves task success over 25 trials per policy. Median task time is computed over successful trials and measures full task execution, not inference latency.

workloads that run many rollouts in parallel, such as large-scale simulation evaluation or reinforcement learning.

Selecting layers outside of the causally determined *action-conditioning interface* hurts performance significantly. Under the same 50,000-update adaptation budget, retaining layer 12 yields only 25.7% Pick success and 20.9% Pick & Place oracle success, compared with 54.7% and 54.0% when retaining layer 24. This contrast supports using the mechanistically identified readout region rather than selecting an arbitrary single-layer interface. Together, these results show that the localized action-conditioning pathway can guide both the design of a new policy and the simplification of an existing one with large efficiency gains, which we now show transfer to real-world deployment.

4.2 $\pi_{0.5}$ IN THE REAL WORLD

We now test whether this design principle transfers to a different architecture in a physical embodiment. For $\pi_{0.5}$, we restrict action-expert attention to layers 14–16, masking this conditioning pathway at the remaining layers. We evaluate both policies on a fruit-in-basket task, in which the robot places three pieces of fruit from the table into a basket. As shown in Table 2, both the original and restricted-interface policies succeed in 22 of 25 trials. The efficiency gain is much smaller than for MolmoBot. Because $\pi_{0.5}$'s action expert reads prefix keys and values that the VLM has already computed, the restriction removes only about 5% of the policy's computation (Section B). The attention mask used on the robot skips no computation at all, and even removing the unused keys and values entirely saves only 4–5% at large batch sizes. Accordingly, our timing measurements show no meaningful difference in inference time between the two policies. Median task duration among successful trials nonetheless decreases from 53.6 s to 45.4 s, about 15%. Since this cannot come from faster inference, it reflects a difference in how the restricted policy executes the task. Although preliminary, these trials show that substantially restricting the interface preserves task success on hardware.

5 DISCUSSION AND CONCLUSION

We used activation patching with controlled counterfactuals to trace how perception becomes action in three VLAs. Across MolmoBot, MolmoAct2, and $\pi_{0.5}$, the picture is consistent. Image tokens contain the scene layout in early layers and the target noun carries which object to pick. In deeper layers, the target's location emerges in text and readout position, as well as image positions in $\pi_{0.5}$, whose bidirectional attention lets them read the instruction. The action expert then reads this information through the *action-conditioning interface*, a narrow band of one to three middle-to-late layers. We then apply these findings to improve the efficiency of the policy. Keeping only the interface layers in MolmoBot cuts theoretical full-policy computation and measured latency by 33%, and raises batched throughput by 44%, while task success remains comparable to the dense policy. More broadly, these results suggest that mechanistic analysis can guide how robot policies are built, not only explain them after training.

Limitations. Our activation-patching experiments use the start of an episode where relevant objects are still in view. This is the only point where clean counterfactuals can be built because both objects are visible, the robot starts in the same state. Therefore our causal map may miss computation that matters only later for example during grasping. The closed-loop evaluations partly address this, as all the policies run in closed loop through complete tasks and remain close to the dense policy in terms of success rate. The real-robot evaluation is also limited and we plan to scale it up in the future. The 25 trials per policy show that restricting attention to the *action-conditioning interface* does not break $\pi_{0.5}$ on hardware.

AI USE STATEMENT

Generative AI tools were used to assist with language editing, LATEX preparation and to propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods. The authors reviewed all AI-assisted work and take responsibility for the final content, claims, and artifacts in this paper.

REPRODUCIBILITY STATEMENT

We will release our code at the time of publication. All three models we study (MolmoBot, MolmoAct2, and $\pi_{0.5}$) are openly available, as are the MolmoBot training data and the MolmoSpaces simulator. The patching protocol, counterfactual comparisons, and trajectory-shift metric are defined in Section 2. The patching experiments need only forward passes and run on a single desktop-class GPU.

REFERENCES

Beyer, L., Steiner, A., Pinto, A. S., et al. PaliGemma: A versatile 3B VLM for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

Bhardwaj, A., Duan, E. W., Dan, P., Ma, W.-C., and Culbertson, P. Decoding task progress from vla representations. *arXiv preprint arXiv:2608.13474*, 2026.

Black, K., Brown, N., Driess, D., Esmail, A., et al. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*, 2025. arXiv:2410.24164.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Brohan, A., Brown, N., Carbajal, J., et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning (CoRL)*, 2023. arXiv:2307.15818.

Buurmeijer, H., Alonso, C. A., Swann, A., and Pavone, M. Observing and controlling features in vision-language-action models. *arXiv preprint arXiv:2603.05487*, 2026.

Chi, C., Feng, S., Du, Y., et al. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems (RSS)*, 2023. arXiv:2303.04137.

Deitke, M., Clark, C., Lee, S., Tripathi, R., Yang, Y., Park, J. S., Salehi, M., Muennighoff, N., Lo, K., Soldaini, L., et al. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 91–104. IEEE, 2025.

Deshpande, A., Guru, M., Hendrix, R., Jauhri, S., et al. MolmoBot: Large-scale simulation enables zero-shot manipulation. *arXiv preprint arXiv:2603.16861*, 2026.

Driess, D., Xia, F., Sajjadi, M. S., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

Fang, H., Duan, J., Clay, D., Wang, S., et al. MolmoAct2: Action reasoning models for real-world deployment. *arXiv preprint arXiv:2605.02881*, 2026.

Geiger, A., Lu, H., Icard, T., and Potts, C. Causal abstractions of neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. arXiv:2106.02997.

Geiger, A., Ibeling, D., Zur, A., Chaudhary, M., Chauhan, S., Huang, J., Arora, A., Wu, Z., Goodman, N., Potts, C., and Icard, T. Causal abstraction: A theoretical foundation for mechanistic interpretability. *Journal of Machine Learning Research*, 26(83):1–64, 2025.

Geva, M., Bastings, J., Filippova, K., and Globerson, A. Dissecting recall of factual associations in auto-regressive language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12216–12235, 2023.

Goyal, A. The hitchhiker's guide to mechanistic interpretability of vision-language-action models. In *Mechanistic Interpretability Workshop at ICML 2026*, 2026.

Grant, B., Zhao, X., and Wang, P. Not all features are created equal: A mechanistic study of Vision-Language-Action models. *arXiv preprint arXiv:2603.19233*, 2026.

Häon, B., Stocking, K. C., Chuang, I., and Tomlin, C. Mechanistic interpretability for steering vision-language-action models. In *Conference on Robot Learning (CoRL)*, 2025. arXiv:2509.00328.

Heimersheim, S. and Nanda, N. How to use and interpret activation patching. *arXiv preprint arXiv:2404.15255*, 2024.

Khan, M. A., Boskov, N., Anwar, F. M., and Khan, M. A. Controlling vision–language–action policies through sparse latent directions. In *Mechanistic Interpretability Workshop at NeurIPS 2025*, 2025.

Khazatsky, A., Pertsch, K., Nair, S., et al. DROID: A large-scale in-the-wild robot manipulation dataset. In *Robotics: Science and Systems (RSS)*, 2024. arXiv:2403.12945.

Kim, M. J., Pertsch, K., Karamcheti, S., et al. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. arXiv:2202.05262.

Molinari, M., Nevali, L., Navani, S., and Younis, O. G. Emergent world representations in openvla. *arXiv preprint arXiv:2509.24559*, 2025.

Mueller, A., Brinkmann, J., Li, M., Marks, S., Pal, K., Prakash, N., Rager, C., Sankaranarayanan, A., Sen Sharma, A., Sun, J., Todd, E., Bau, D., and Belinkov, Y. The quest for the right mediator: Surveying mechanistic interpretability for NLP through the lens of causal mediation analysis. *Computational Linguistics*, 52(1):331–378, March 2026. doi: 10.1162/coli.a.572. URL https://aclanthology.org/2026.cl-1.10/.

Nguyen, G.-B. et al. Finetuning vision-language-action models requires fewer layers than you think. *arXiv preprint arXiv:2606.20246*, 2026.

NVIDIA, Bjorck, J., Castañeda, F., et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Physical Intelligence, Black, K., Brown, N., Darpinian, J., et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Shi, H. et al. VLA-Trace: Diagnosing vision-language-action models through representation and behavior tracing. *arXiv preprint arXiv:2605.30117*, 2026.

Shukor, M., Aubakirova, D., Capuano, F., et al. SmolVLA: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Sun, G. et al. Drop-then-recovery: How redundant are vision-language-action models? *arXiv preprint arXiv:2606.27755*, 2026.

Swann, A., McGranahan, L., Buurmeijer, H., Kennedy III, M., and Schwager, M. Sparse autoencoders reveal interpretable and steerable features in VLA models. *arXiv preprint arXiv:2603.19183*, 2026.

Vig, J., Gehrmann, S., Belinkov, Y., et al. Causal mediation analysis for interpreting neural NLP: The case of gender bias. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. arXiv:2004.12265.

Wang, K., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the wild: A circuit for indirect object identification in GPT-2 small. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2211.00593.

Yang, T. and Mokalla, A. Interpretable signals for vision-language-action model failure. In *Mechanistic Interpretability Workshop at ICML 2026*, 2026.

Yang, Y. et al. EfficientVLA: Training-free acceleration and compression for vision-language-action models. *arXiv preprint arXiv:2506.10100*, 2025.

Yue, Y. et al. DeeR-VLA: Dynamic inference of multimodal large language models for efficient robot execution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2411.02359.

Zhang, F. and Nanda, N. Towards best practices of activation patching in language models: Metrics and methods. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2309.16042.

Zhang, F. et al. Revisiting parameter redundancy in vision-language-action models: Insights from VLM-to-VLA adaptation. *arXiv preprint arXiv:2606.31382*, 2026a.

Zhang, K. et al. A1: A fully transparent open-source, adaptive and efficient truncated vision-language-action model. *arXiv preprint arXiv:2604.05672*, 2026b.

Zhang, R. et al. MoLe-VLA: Dynamic layer-skipping vision language action model via mixture-of-layers for efficient robot manipulation. *arXiv preprint arXiv:2503.20384*, 2025.

A Additional Results

A.1 Backbone Patching in MolmoBot

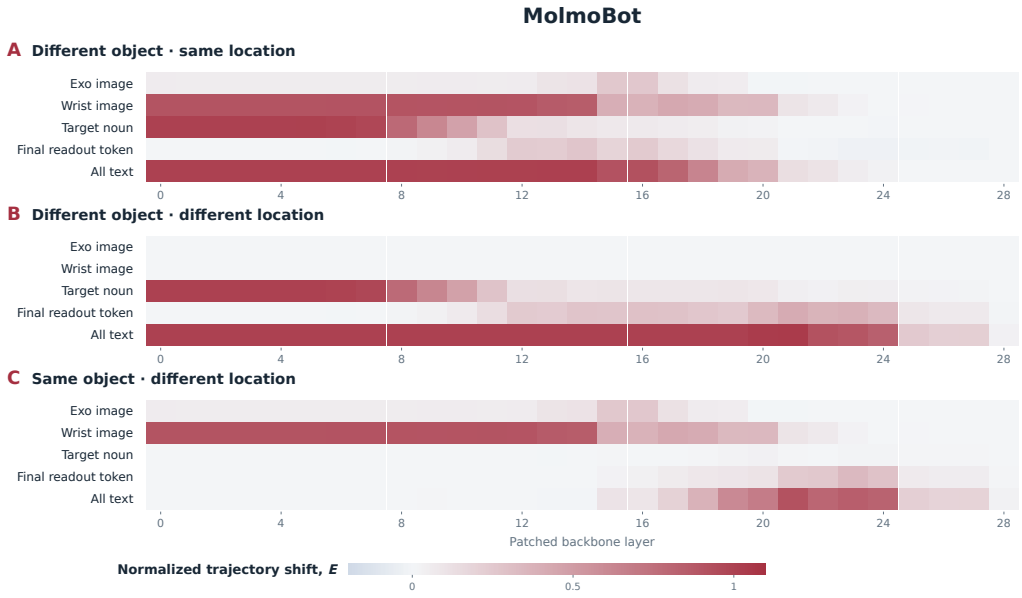


Figure 5: **MolmoBot backbone patching across 13 object pairs.** Same comparisons, token groups, and metric as Figure 4. Each cell averages 416 patches: 26 donor–recipient pairs, each patched in both directions under eight noise seeds. Image patches in B are exactly zero because MolmoBot's image positions cannot attend to the instruction. Omitted layers 29–35 have $|E| < 0.005$.

Figure 5 shows backbone patching for MolmoBot on the same 13 object pairs as Figure 4. We see that its causal structure closely matches MolmoAct2's. In both models, the target noun carries object identity in early layers (A and B). Location is then distributed later across text positions (C). In both MolmoACT2 and MolmoBot, image patches have no effect in B, because image positions precede the instruction and cannot attend to it. The main difference we see visually separating the two models is which camera matters: the wrist view dominates in MolmoBot, and the exterior view dominates in MolmoAct2.

This similarity between the models causal structure holds even though the two models are trained on completely different data (simulated versus real demonstrations) and their action experts attend to the backbone differently. In $\pi_{0.5}$, whose backbone attends bidirectionally over the prefix image tokens absorb some instruction conditioned information and causal influence (Figure 4). Together, the three models suggest that the backbone architecture, and in particular its attention mask, shapes where task information lives in a VLA.

A.2 Text Tokens That Geometric Location in MolmoAct2

Based on the backbone patching results, particularity with respect to geometry, we know that the policy represents information about the target object position in mid-to-late layer text tokens. Geometric information is distributed across many different tokens. To identify which text positions carry this location, we search for the smallest set of tokens which jointly patch recovers the effect of patching all of the text. The search runs over all 52 comparison-C donor–recipient pairs and the results for MolmoAct2 are shown in Figure 6.

At layer 18, four tokens suffice: the target noun, the period that ends the task sentence, the "The" that follows it, and the "the" in "the current state of the robot". Together they keep 89% of the effect. At layers 19–21 the sets grow to 10–11 tokens that add `<action_output>` and words of the action question (and, at layer 21, `<control_start>` and `assistant`), keeping 76–77%. At layer 22 the action question drops out, and `<action_output>` with the four sentence-boundary tokens keeps 77%.



Figure 6: **The smallest token sets that carry location in MolmoAct2, across all 13 object pairs.** For each layer, filled cells mark the smallest set of text tokens, found by greedy forward selection, whose joint patch keeps at least 75% of the effect of patching all 79 non-state text tokens in comparison C; the right column gives the number of tokens. Effects are averaged over all 52 comparison-C donor–recipient pairs with equal weight per object pair. The sets keep 76–89% of the effect on the noise seed used for selection and 72–90% on three held-out seeds. The target noun, which differs across object pairs, is labeled `<object>`; state tokens were not searched.

Removing any single token from any set drops retention below 75%. Location is therefore carried jointly by a few structural positions (sentence boundaries, function words, and the action question) before it concentrates on the readout token. We hypothesize this is where the computation of the target object location is happening.

B Compute and Latency

Here we provide more details on our theoretical FLOP calculations and empirical latency and throughput results for MolmoBot. One MolmoBot action chunk consists of one vision-encoder pass, one VLM pass, and ten action-expert denoising passes. Restructuring the cross-attention blocks also allows us to terminate the VLM computation early as nothing reads VLM layers 25–35.

Estimated operation counts. We assume that a dense layer with N parameters applied to T tokens costs about $2NT$ FLOPs, adding attention products and ignoring normalization, softmax, and embeddings. A VLM layer which for Molmobot is ≈101M parameters and 928 tokens costs 0.20 TFLOPs. Therefore running only 25 out of 36 layers reduces the VLM computation from 7.25 to 5.04 TFLOPs. A cross-attention block costs 5.9 GFLOPs per flow step, the majority of it from projecting the 928 context tokens into the expert. 36 blocks over ten steps cost 2.13 TFLOPs while the rest of the action expert only costs 0.08 TFLOPs. Therefore keeping one block reduces the expert from 2.22 to 0.14 TFLOPs. With the vision encoder fixed at 2.46 TFLOPs, the full policy falls from 11.93 to 7.64 TFLOPs per action chunk overall. We apply the same calculations to the 36 layer policy and L23-25 policy and show the results in Table 3

For $\pi_{0.5}$ the saving is small because its action expert reads prefix keys and values that the VLM has already computed. Most of the gain comes from letting the VLM stop after layer 16 (0.23 TFLOPs). The full policy falls from 4.76 to 4.52 TFLOPs (-5%). Given how small these estimated saving are, we do not empirically analyze $\pi_{0.5}$'s latency or throughput.

	Vision	VLM	Action expert	Total	Change
MolmoBot, all 36 layers	2.46	7.25	2.22	11.93	–
MolmoBot, layer 24	2.46	5.04	0.14	7.64	-36%
MolmoBot, layers 23–25	2.46	5.24	0.26	7.96	-33%
$\pi_{0.5}$, all 18 layers	0.66	3.98	0.13	4.76	–
$\pi_{0.5}$, layers 14–16	0.66	3.75	0.11	4.52	-5%

Table 3: Estimated TFLOPs per action chunk. Restricted policies stop the VLM after the deepest layer the action expert reads.

Measured latency and throughput. We timed MolmoBot on a single H100 in bfloat16 with ten flow-matching steps and two cameras $\times$ two frames (Table 4). At batch size 1, the saving comes almost entirely from the action expert. At batch size 64, the all-layer expert's 36 cross-attention blocks become compute-bound, so the throughput gain is larger. The measured latency reduction (35%) matches the estimated full-policy reduction (36%).

MolmoBot	Latency (ms)	Throughput (chunks/s)	Memory (GB)
All 36 layers	311 ± 3	21.5	43
Layer 24	204 ± 16 (-35%)	31.8 ($+48\%$)	32
Layers 23–25	209 ± 2 (-33%)	30.9 ($+44\%$)	–

Table 4: Measured inference cost on one H100 (bfloat16, ten flow-matching steps). Latency is at batch size 1; throughput and peak memory are at batch size 64. Restricted policies stop the VLM after the deepest layer the action expert reads.

C Implementation Details

MolmoBot evaluation. All MolmoBot results use the evaluation code released with MolmoBot (Deshpande et al., 2026). We evaluate on three fixed sets of 1,000 episodes (Pick, Pick & Place, Pick & Place Next-To) in held-out ProcTHOR-Objaverse houses, run in the MolmoSpaces simulator with the released configuration. Every policy is evaluated the same set of episodes with identical noise seeds.

Model Training We use the released MolmoBot training code with its two-stage recipe, except modifying the set of action-expert blocks that keep cross-attention. Stage 1 initialises from Molmo2-4B and trains for 200K steps on single-frame observations then stage 2 continues for another 50K steps with two of horizon frames. The *adapt* variants start from the released MolmoBot-DROID checkpoint and then run the stage-2 recipe for 50K steps. For $\pi_{0.5}$ both policies are fine-tuned from the released $\pi_{0.5}$ base checkpoint (Physical Intelligence et al., 2025) on DROID (Khazatsky et al., 2024) with the published openpi DROID recipe.