DROP-THEN-RECOVERY: HOW REDUNDANT ARE VISION-LANGUAGE-ACTION MODELS?

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models enable instruction-driven robotic manipulation, but they inherit oversized language backbones from pretrained VLMs whose capacity far exceeds what is needed for short robotic instructions. This raises a basic question: how much of a VLA model is actually necessary for closed-loop control? In this work, we study architectural redundancy in VLA models by using transformer block removal as a controlled intervention. We introduce **Drop-Then-Recovery (DTR)**, an analysis protocol that removes selected blocks from a pretrained VLA model and then fine-tunes the resulting model to measure whether the removed capacity was necessary for downstream control. To make this intervention reliable, we propose **GateProbe**, a one-shot virtual-gate sensitivity metric that ranks blocks by their contribution to the downstream action loss. Across multiple VLA architectures, manipulation benchmarks and even real-robot industrial scenarios, we find a strong asymmetry in post-removal recoverability: *language backbones are highly redundant for standard robotic manipulation tasks, whereas vision and action pathways are substantially less tolerant to removal*. On LIBERO, removing half of the LLM blocks even improves OpenVLA-OFT from 95.0% to 98.3% under the same downstream fine-tuning budget, and retaining only two language blocks still recovers baseline-level performance. These results suggest that current VLA benchmarks may exert limited pressure on deep language grounding and compositional instruction understanding, and that future VLA architectures should allocate capacity more deliberately across language, vision, and action components. The code will be released upon acceptance of the paper.

1 INTRODUCTION

Vision-language-action (VLA) models have become a common framework for instruction-driven robotic manipulation (Zitkovich et al., 2023; Kim et al., 2025b; Black et al., 2025b). Given visual observations and a natural-language instruction, a VLA policy directly predicts robot actions. Recent systems achieve strong results by combining pretrained vision-language backbones with action prediction modules (Black et al., 2025a; Gemini Robotics Team et al., 2025; Kim et al., 2025a).

A key design choice in these models is to inherit large pretrained backbones. While this provides broad visual and linguistic knowledge (Open X-Embodiment Collaboration et al., 2023; Kim et al., 2025b; Black et al., 2025b), it may be excessive for standard manipulation benchmarks, where instructions are often short and templated, such as "*pick up the red cup*" (Liu et al., 2023; Fei et al., 2025; Chen et al., 2025a). This raises a basic question: **how much of a VLA model is actually needed for closed-loop robotic control?**

This question cannot be answered from parameter count alone. A language block useful for web-scale vision-language modeling may be unnecessary for a downstream control task, while a small action module may be critical because action errors can accumulate over long horizons. Crucially, redundancy must be measured by closed-loop task success after recovery fine-tuning, not merely by single-step prediction loss.

In this work, we study VLA redundancy through **Drop-Then-Recovery (DTR)**, a controlled protocol that removes transformer blocks and then fine-tunes the remaining model on the downstream task. The goal of DTR is not only to produce a smaller model, but also to measure whether the removed capacity was needed for robotic control: if a dropped model recovers its task success, the removed

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.

blocks were not essential for the evaluated task distribution. A central challenge is deciding which blocks to remove: existing layer-dropping metrics often rely on static similarity, parameter magnitude, or immediate degradation (Men et al., 2025; Gromov et al., 2025; Song et al., 2024; He et al., 2024), but do not necessarily predict *recoverability*. We therefore propose **GateProbe**, a one-shot virtual-gate metric that ranks blocks by their effect on the downstream action loss.

Our experiments reveal a clear pattern across VLA architectures and manipulation benchmarks: **the language backbone is highly redundant under current standard manipulation benchmarks, while the action pathway is much less tolerant to removal**. On LIBERO, dropping half of the LLM blocks already surpasses the full-model baseline under matched training compute, *e.g.*, OpenVLA-OFT reaches 98.3% vs. 95.0% and $\pi_{0.5}$ reaches 94.0% vs. 91.7%. Even retaining only two language blocks still matches baseline performance (OpenVLA-OFT 95.1%; $\pi_{0.5}$ 91.0%). In addition to the strong performance of structured Dropping on VLA models, this finding suggests that standard manipulation benchmarks may not fully test language grounding or compositional reasoning, pointing to the need for both more capacity-balanced VLA designs and more linguistically demanding benchmarks. Our contributions are summarized:

- **VLA language backbones are highly over-sized for current manipulation benchmarks.** Across multiple VLA architectures, we find that most language blocks can be removed and recovered with little or no loss in task success.
- **DTR provides a simple way to measure and use this redundancy.** We study redundancy by physically removing transformer blocks from pretrained VLA models and then fine-tuning the smaller models. This protocol tests whether the removed capacity was needed for closed-loop control, while also producing a smaller dense model when the removed blocks are recoverable.
- **GateProbe improves block selection under aggressive removal.** We propose a one-shot virtual-gate metric that ranks blocks by their effect on the downstream action loss. Compared with static metrics, GateProbe better selects recoverable block sets, especially when only a few language blocks are kept.
- **Current VLA benchmarks may under-test language grounding.** The ease of recovering from large language-block removal suggests that standard manipulation benchmarks may not require rich language understanding. Our results motivate benchmarks with more compositional instructions, stronger language grounding, and longer-horizon language-conditioned control.

2 Related work

Vision-language-action models. VLA models combine pretrained vision-language backbones with action prediction modules for instruction-driven robotic manipulation (Zitkovich et al., 2023; Kim et al., 2025b; Black et al., 2025b), and recent scaling of both data and backbone capacity has led to strong performance across diverse tasks (Black et al., 2025a; Kim et al., 2025a).

Model compression for LLMs and VLMs. Compression techniques for LLMs and VLMs include post-training quantization (Frantar et al., 2023), unstructured or semi-structured pruning (Sun et al., 2023; Frantar & Alistarh, 2023), structured pruning (Ma et al., 2023), and transformer block removal based on similarity or sublayer-importance criteria (Men et al., 2025; Gromov et al., 2025; Song et al., 2024; He et al., 2024). These methods typically evaluate compressed models *without* recovery fine-tuning. While tolerable for language or vision-language benchmarks, this drop-only regime is insufficient for VLA models, where small action errors compound over long horizons and cause task-level collapse. Our work therefore studies the *compress-then-recover* regime, where post-compression fine-tuning is essential for judging what capacity is truly necessary.

Compression and efficiency for VLA models. Recent work improves VLA efficiency through quantization, token pruning, layer skipping, distillation and so on (Xu et al., 2026; Wang et al., 2025b; Yang et al., 2025; Wang et al., 2025a; Zhang et al., 2025; Chen et al., 2025b; Jeon et al., 2026). Concurrent studies show that naive pruning can harm VLA behavior and that redundancy is asymmetric across components (Jabbour et al., 2025; Grant et al., 2026). These efforts confirm that VLA redundancy exists, but *how redundant* each component is and *where the recoverability limit lies* remain open. We use structured block removal as a controlled probe and pair it with **`GateProbe`**, a recoverability-aware importance metric, to systematically evaluate redundancy in VLA models.

3 METHOD

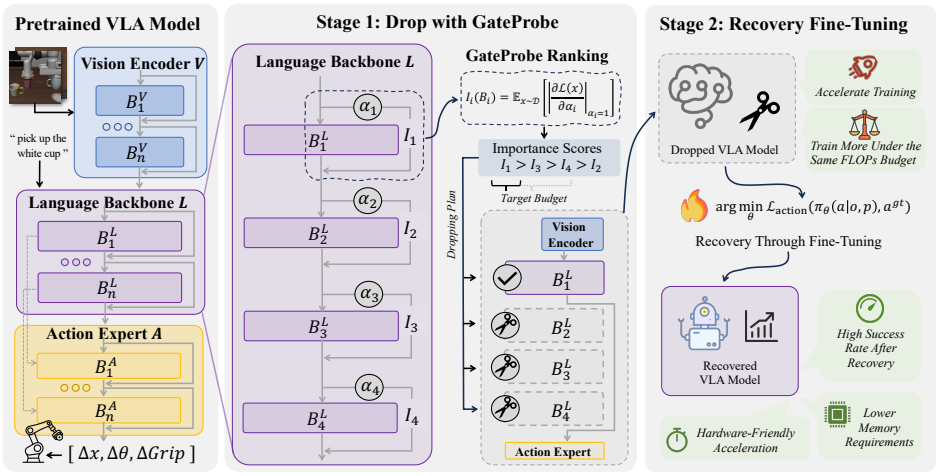


Figure 1: Overview of DTR. A pretrained VLA model's transformer blocks are ranked by importance, the least important are physically removed, and the smaller model is recovery fine-tuned.

A VLA model consists of a vision encoder $\mathcal{V}$, a language backbone $\mathcal{L}$, and an action head $\mathcal{A}$, each built from stacked transformer blocks with residual connections. We denote the full set of droppable blocks across all components as

$$\mathcal{B} = \underbrace{\{B_1^{\mathcal{V}}, \ldots, B_{N_V}^{\mathcal{V}}\}}_{\text{vision}} \cup \underbrace{\{B_1^{\mathcal{L}}, \ldots, B_{N_L}^{\mathcal{L}}\}}_{\text{language}} \cup \underbrace{\{B_1^{\mathcal{A}}, \ldots, B_{N_A}^{\mathcal{A}}\}}_{\text{action}}, \tag{1}$$

where each block B_i follows the residual form $h_i = h_{i-1} + F_i(h_{i-1};\, \theta_i)$. Note that in dual-stream architectures like $\pi_{0.5}$ (Black et al., 2025a), dropping a language block does not remove all of its parameters (K/V projections are retained for cross-attention). We detail this mechanism and its impact on compression ratios in Appendix B.

3.1 THE DROP-THEN-RECOVERY (DTR) PROTOCOL

We use block removal as a structural probe for VLA redundancy. Unlike weight-level compression methods such as quantization or pruning, block removal operates on explicit architectural units while leaving the remaining network as a dense model. This makes the intervention easy to compare across components and less tied to hardware-specific kernels. **DTR** formalizes this idea as an analysis protocol that first removes selected transformer blocks and then measures whether the resulting model can recover task performance after fine-tuning. **DTR** operates in two stages (Figure 1):

Stage 1: Drop. Given an importance metric I and a target drop count K, we select and physically remove the K least important blocks:

$$\mathcal{S} = \operatorname{argsort}_K \big\{ I(B_i) \big\}_{B_i \in \mathcal{B}}, \qquad \mathcal{M}_{\text{drop}} = \mathcal{M} \setminus \mathcal{S}. \tag{2}$$

Dropping short-circuits the residual ($h_i = h_{i-1}$) and discards θ_i, producing a genuinely smaller dense model with proportionally reduced FLOPs, memory, and latency on any hardware. Since dropping occurs *before* recovery, the smaller model also trains faster and uses less GPU memory.

Stage 2: Recovery. The dropped model is fine-tuned on the downstream task:

$$\theta^* = \arg\min_{\theta} \; \mathcal{L}_{\text{action}}\big(\pi_\theta(a \mid o, p),\; a^{\text{gt}}\big)\,, \tag{3}$$

where o is the observation, p is the language instruction, a^{gt} is the demonstration action, and $\mathcal{L}_{\text{action}}$ is the action prediction loss (*e.g.*, MSE for continuous actions, flow-matching for diffusion-based heads). Recovery is critical because in closed-loop control, even small degradations in action quality can compound into task failures over long horizons.

Recoverability. We refer to the task performance of a dropped model *after* recovery fine-tuning as its **recoverability**. This is distinct from *importance* (how much performance drops immediately upon removal): a block may cause a large zero-shot degradation yet be easily recoverable after fine-tuning.

3.2 BLOCK SELECTION VIA GATEPROBE

A critical component of **DTR** is the importance metric $I(B_i)$ used to decide which blocks to drop (Eq. 2). Existing static metrics such as cosine similarity, perplexity, and magnitude (Section 2) are computed from weights or forward representations without downstream gradients; they only measure the immediate performance after dropping, without capturing a block's *recovery potential* after fine-tuning. Gradient-based metrics like Taylor sensitivity estimate recoverability better but degrade under extreme compression ratios, while second-order or ablation-style alternatives are computationally expensive. To address both limitations, we propose **GateProbe**.

GateProbe: virtual gate sensitivity. **GateProbe** is a metric that operates in *activation space* and directly measures the sensitivity of the task loss to each block's *functional contribution*. The key idea is to introduce a virtual scalar gate α_i on each block's residual branch:

$$\tilde{h}_i = h_{i-1} + \alpha_i \cdot F_i(h_{i-1};\, \theta_i), \qquad \alpha_i = 1. \tag{4}$$

Setting $\alpha_i = 0$ is equivalent to dropping block B_i; setting $\alpha_i = 1$ recovers the original model. The **GateProbe** importance score is the expected absolute sensitivity of the task loss to this gate:

$$I_{\text{gate}}(B_i) = \mathbb{E}_{x\sim\mathcal{D}}\left[\left|\frac{\partial \mathcal{L}(x)}{\partial \alpha_i}\right|_{\alpha_i=1}\right]. \tag{5}$$

By the chain rule, this can be computed without explicitly introducing α_i into the model:

$$\left.\frac{\partial \mathcal{L}}{\partial \alpha_i}\right|_{\alpha_i=1} = \left\langle \frac{\partial \mathcal{L}}{\partial h_i},\; F_i(h_{i-1}) \right\rangle, \tag{6}$$

where $\mathcal{L}$ denotes the downstream action loss $\mathcal{L}_{\text{action}}$ of Eq. 3, $\mathcal{D}$ a small calibration set drawn from the downstream data, $\partial\mathcal{L}/\partial h_i$ the downstream gradient flowing back through all subsequent layers (available from backpropagation), and $F_i(h_{i-1}) = h_i - h_{i-1}$ the block's residual contribution (computed from cached hidden states during the forward pass). The score is thus the inner product of the *downstream gradient* and the *block's residual contribution*, capturing both how much the block changes the representation and how much the downstream computation relies on that change. Further details, including a Taylor-expansion interpretation and the full algorithm, are provided in Appendix D.

4 SIMULATION EXPERIMENTS

4.1 SETUP

Models. We evaluate **DTR** on four representative VLA architectures: $\pi_{0.5}$ (Black et al., 2025a), OpenVLA-OFT (Kim et al., 2025a), Lingbot-VLA (Wu et al., 2026), and GigaBrain-0 (GigaBrain Team et al., 2025). These models cover different backbone families, model scales, and action-head designs, including continuous regression, flow matching, and diffusion-based action prediction. Detailed architecture descriptions are provided in Appendix A.

Benchmarks. We evaluate on three simulation benchmarks: LIBERO (Liu et al., 2023), LIBERO-Plus (Fei et al., 2025), and RoboTwin 2.0 (Chen et al., 2025a). LIBERO is used for the main redundancy and metric studies, while LIBERO-Plus and RoboTwin 2.0 test robustness and cross-benchmark transfer. Training details and hyperparameters are provided in Appendix H.

4.2 WHICH VLA COMPONENT IS MOST REDUNDANT?

The Language backbone is overwhelmingly more redundant than Vision or Action.

To observe VLA redundancy without importance-metric bias, we apply two simple strategies to each component (Vision, Language, Action) independently: Drop Half (removing all odd-indexed blocks) and Keep 2 (retaining only the first and last blocks). Results are shown in Table 1.

Table 1: VLA component redundancy on LIBERO (fixed interventions: Drop Half and Keep 2; vision drop lists in Appendix G). *: OpenVLA-OFT action compression reduces hidden dimension (Appendix C).

Model	Setting	Component	Size	FLOPs	Spatial	Object	Goal	Long	Avg.
OpenVLA -OFT	Baseline	—	100%	100%	97.2	98.4	95.6	88.6	95.0
	Drop Half	Vision	96.4%	95.6%	82.6	99.0	77.2	76.8	83.9
		Language	55.5%	55.0%	**99.0**	**100.0**	**97.8**	**96.4**	**98.3**
		Action*	99.5%	100.0%	92.4	99.0	93.6	92.8	94.5
	Keep 2	Vision	92.2%	91.6%	80.2	97.6	92.4	50.4	80.2
		Language	16.6%	15.6%	**97.2**	99.0	**95.4**	88.6	**95.1**
		Action*	99.3%	99.9%	89.2	**99.2**	76.6	**90.8**	89.0
$\pi_{0.5}$	Baseline	—	100%	100%	96.6	95.0	93.0	82.0	91.7
	Drop Half	Vision	89.5%	93.3%	84.0	90.2	82.0	66.8	80.8
		Language	60.6%	57.9%	**98.8**	**98.6**	93.4	82.4	**93.3**
		Action	93.8%	99.5%	95.8	95.0	**92.4**	**89.6**	93.2
	Keep 2	Vision	79.8%	87.0%	69.4	75.2	62.4	42.6	62.4
		Language	30.0%	25.1%	**94.6**	**96.0**	**90.6**	**82.6**	**91.0**
		Action	88.9%	99.1%	3.6	40.8	16.0	44.4	26.2

Table 2: Effect of dropping granularity on LLM backbone redundancy (Drop Half, LIBERO).

Model	Target	Size	FLOPs	Spatial	Object	Goal	Long	Avg.
OpenVLA -OFT	MHA	85.2%	84.4%	95.0	99.2	**99.2**	76.4	91.9
	MLP	70.3%	70.6%	83.2	96.0	15.4	72.8	65.6
	Block	55.5%	55.0%	**99.0**	**100.0**	97.8	**96.4**	**98.3**
$\pi_{0.5}$	MHA	97.0%	95.3%	96.4	98.0	**95.8**	86.2	**94.1**
	MLP	63.7%	62.5%	96.8	97.4	93.4	**86.8**	93.6
	Block	60.6%	57.9%	**98.8**	**98.6**	93.4	82.4	93.3

On both architectures, Language tolerates compression far better than Vision or Action. On OpenVLA-OFT, Language Drop Half removes 44.5% of parameters while matching or exceeding the baseline SR (98.3% vs. 95.0%), whereas Vision removes only 3.6% but drops to 83.9%. Under extreme compression (Keep 2), Language remains close to baseline, while Vision and Action collapse to 62.4% and 26.2% respectively on $\pi_{0.5}$. We thus focus on language block removal in all subsequent experiments.

4.3 What is the optimal dropping granularity?

Whole-block dropping is the most cost-effective granularity for compression.

We next investigate the optimal dropping granularity within the language backbone, comparing three options: whole blocks, MHA sublayers only, and MLP sublayers only (He et al., 2024; Gromov et al., 2025). Here all three settings remove a fixed half of the backbone, so no importance metric is involved; **`GateProbe`** also applies at the sublayer level, since the virtual gate can be attached to the MHA or MLP branch in the same way as to a whole block. Table 2 shows the results. On OpenVLA-OFT, block dropping (98.3%) substantially outperforms MHA (91.9%) and MLP (65.6%). On $\pi_{0.5}$, all granularities achieve similar SR (93.3% to 94.1%), but block dropping compresses the most. Therefore, we adopt whole-block dropping as the default for all subsequent experiments.

4.4 Which importance metrics predict recoverability?

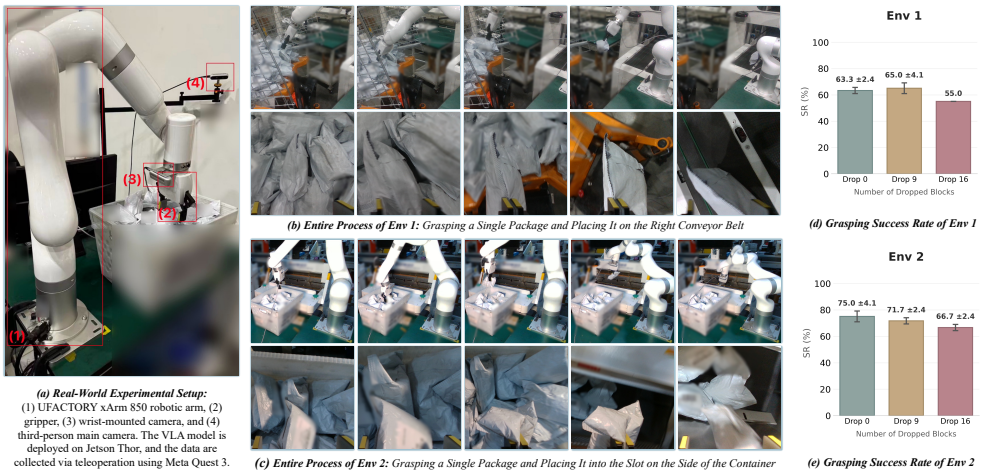
*Gradient-based metrics, including **`GateProbe`**, predict recoverability better than static ones; **`GateProbe`** shows the clearest advantage under aggressive compression.*

Table 3 compares metrics on $\pi_{0.5}$ across four drop levels. Non-gradient metrics (CosSim, Magnitude, PPL) are cheap but consistently underperform across all drop levels. Gradient-based metrics (Taylor, IGIA) work well at moderate compression but degrade under extreme settings. **`GateProbe`**

Table 3: Importance metrics for LLM block dropping on $\pi_{0.5}$ / LIBERO (bsz 32, 30K steps). †: These metrics select the same blocks at this drop level. Please see Table 9 for kept block indices.

Setting	Metric	Spatial	Object	Goal	Long	Avg.
Baseline	—	96.6	95.0	93.0	82.0	91.7
Drop-9 (9/18)	Taylor / IGIA†	97.0	97.0	**94.0**	88.6	**94.2**
	GateProbe	97.0	98.2	92.0	**88.8**	94.0
	Fisher	**97.2**	94.4	91.0	84.4	91.8
	Hessian	96.0	95.6	89.6	84.0	91.3
	CosSim	93.8	**99.0**	89.6	78.6	90.2
	PPL	93.0	97.6	90.0	78.0	89.6
	CosSim (contig.)	94.6	94.8	86.4	76.4	88.0
	Magnitude	95.2	94.6	89.8	72.4	88.0
Drop-12 (12/18)	**GateProbe**	**96.8**	97.4	90.6	83.6	**92.1**
	Taylor	95.4	97.0	90.0	**85.8**	92.0
	IGIA	**96.8**	97.2	91.0	81.6	91.6
	Fisher	95.0	97.6	88.8	81.0	90.6
	CosSim (contig.)	94.6	96.4	**91.6**	78.2	90.2
	Hessian	93.4	95.4	90.2	81.4	90.1
	CosSim	95.6	97.4	87.6	77.0	89.4
	PPL	94.0	**97.8**	90.6	71.2	88.4
	Magnitude	94.8	97.0	86.0	74.2	88.0
Drop-16 (16/18)	**GateProbe**/ Fisher†	**96.6**	96.8	**90.6**	**84.6**	**92.2**
	Hessian	94.6	96.2	89.6	72.8	88.3
	PPL	92.2	94.2	88.8	73.6	87.2
	Taylor	92.6	**97.6**	80.2	70.6	85.2
	IGIA	93.8	88.6	84.4	70.0	84.2
	Mag. / CosSim / CosSim (c.)†	94.6	97.0	73.2	62.6	81.9
Drop-17 (17/18)	**GateProbe**/ Fisher / Hessian / IGIA / PPL†	**94.4**	**97.2**	**88.2**	**75.0**	**88.7**
	Taylor	91.6	94.4	79.2	72.2	84.4
	Mag. / CosSim / CosSim (c.)†	92.2	90.6	82.0	70.2	83.8

achieves the best or second-best at all four levels, with a growing advantage under aggressive compression (+3.9 at Drop-16, +4.3 at Drop-17). We use **GateProbe** as the default in all subsequent experiments; note that the component-level redundancy findings (Sections 4.2 and 4.3) use fixed, metric-independent interventions and do not depend on this choice. Details of each metric and the kept block indices are provided in Appendix E.



(a) Real-World Experimental Setup: (1) UFACTORY xArm 850 robotic arm, (2) gripper, (3) wrist-mounted camera, and (4) third-person main camera. The VLA model is deployed on Jetson Thor, and the data are collected via teleoperation using Meta Quest 3.

(b) Entire Process of Env 1: Grasping a Single Package and Placing It on the Right Conveyor Belt

(c) Entire Process of Env 2: Grasping a Single Package and Placing It into the Slot on the Side of the Container

(d) Grasping Success Rate of Env 1

(e) Grasping Success Rate of Env 2

Figure 2: Real-world experimental setup and main results.

5 REAL-WORLD EXPERIMENTS

Language-backbone redundancy transfers from simulation to real-world deployment: dropped models perform comparably to the full model in near-training conditions, but exhibit reduced robustness under visual and physical perturbations.

Platform and data. We deploy $\pi_{0.5}$ on a UFACTORY xArm 850 equipped with a UFACTORY xArm Gripper G2, a wrist-mounted Intel RealSense D435 camera, and a third-person camera (Figure 2-a). The model runs on an NVIDIA Jetson Thor. Training data are collected via teleoperation with Meta Quest 3 at 10 Hz, totaling ∼110K frames (∼600 grasps). Training follows the setting in Table 4.

Task. The target scenario is warehouse parcel sorting: the robot picks soft-body packages from a filled container and places them onto a conveyor or into adjacent slots. The task poses several challenges: (i) packages are tightly stacked and deformable, so boundaries are visually ambiguous; (ii) each package contains a medicine bottle internally, leaving only a limited graspable area on the surface; and (iii) each attempt must grasp exactly one package. We test two layouts: **Env 1** picks from a wire-mesh container to a right-side conveyor (Figure 2-b), and **Env 2** picks from a box to side slots (Figure 2-c).

Main results. Figure 2-d, e reports success rates averaged over 3 runs × 20 grasps. In Env 1, Drop-9 (65.0%) slightly exceeds the full model (63.3%), while Drop-16 degrades to 55.0%. In Env 2, the full model achieves 75.0%, with Drop-9 at 71.7% and Drop-16 at 66.7%. This mirrors the simulation pattern: removing half the language blocks preserves task performance, while retaining only 2 of 18 blocks causes moderate degradation.

Robustness under distribution shift. As shown in Figure 3, we further evaluate under six out-of-distribution (OOD) conditions with a single run of 20 grasps in Env 2, corresponding to Figure 2-c: three lighting changes (pink, green, flashing), novel objects, altered container orientation, and container removal. Under mild perturbations such as container orientation, dropped models remain close to the full model (75%/70%/70% for Drop-0/9/16). However, under stronger perturbations the performance gap becomes evident: for example, under green light Drop-16 falls to 35% (vs. 50% for the full model), and under container removal Drop-16 drops to 40% (vs. 60%). We provide a more detailed analysis of robustness degradation under perturbations in Section 6.2.

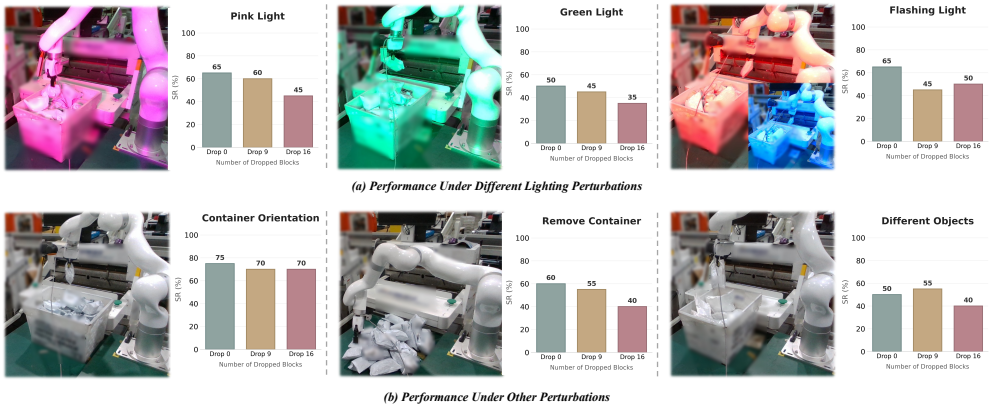


Figure 3: Robustness under distribution shift. (a) Lighting perturbations. (b) Physical perturbations.

6 ANALYSIS AND DISCUSSION

6.1 WHAT ARE THE PRACTICAL BENEFITS OF REDUCING REDUNDANCY?

Removing redundant blocks yields both higher training throughput and hardware-agnostic inference acceleration, orthogonal to existing compression methods.

The redundancy identified above has two direct practical consequences.

Higher training throughput under fixed compute. Since DTR removes blocks before fine-tuning, a dropped model is cheaper per step, allowing more training iterations under the same compute budget. We verify this by matching total FLOPs to the baseline (bsz 32, 30K steps) through scaled batch

Table 4: FLOPs-matched comparison on $\pi_{0.5}$ / LIBERO (baseline: bsz 32 $\times$ 30K steps).

Setting	Size	FLOPs	Bsz	Steps	Spatial	Object	Goal	Long	Avg.
Baseline	100%	100%	32	30K	96.6	95.0	93.0	82.0	91.7
Drop-9	60.6%	57.9%	64	25.9K	92.8	97.2	91.0	**88.2**	92.3
Drop-12	47.5%	43.8%	64	34.2K	**96.8**	**98.4**	94.4	85.0	**93.7**
Drop-16	30.0%	25.1%	64	59.8K	95.2	95.6	**95.2**	84.2	92.6
Drop-17	25.6%	20.4%	64	73.5K	**96.8**	95.8	92.4	78.8	91.0

Table 5: DTR vs. recovery-free dropping on OpenVLA-OFT / LIBERO-Goal. Task Speedup = Act. Speedup / Step Ratio. We report the latency (ms) and memory (GB) for generating a single action.

Method	SR (%) ↑	Act. Speedup ↑	Latency ↓	Memory ↓	Step Ratio ↓	Task Speedup ↑
Dense Baseline	98.0	1.00×	225.1	14.40	1.00×	1.00×
DTR-16	**100.0**	1.56×	144.4	8.36	**0.95**×	1.64×
DTR-30	90.0	**2.94**×	**76.7**	**3.06**	1.13×	**2.60**×
CosSim zero-shot drop (applied to fine-tuned baseline, no recovery)						
Attn Drop 4	**100.0**	1.09×	206.1	13.90	0.99×	1.10×
Attn Drop 8	98.0	1.18×	191.5	13.40	1.08×	1.09×
Block Drop 4	78.0	1.05×	214.4	12.89	1.46×	0.72×
Block Drop 8	18.0	1.31×	171.5	11.38	2.39×	0.55×

size and steps (Table 4). Drop-9 through Drop-16 all match or exceed the baseline, with Drop-12 achieving the best average (93.7%, +2.0). Even Drop-17 (a single language block) recovers to 91.0%.

Hardware-agnostic inference acceleration. Unlike quantization (which requires low-bit kernel support) and sparse pruning (which requires structured sparsity patterns), DTR produces a standard dense model with fewer layers. The resulting speedup applies to any hardware without specialized kernels or runtime support (see Appendix L).

Table 5 benchmarks structural dropping methods on OpenVLA-OFT / LIBERO-Goal. We compare DTR (drop on base model, then fine-tune) with zero-shot CosSim-based dropping (drop blocks directly from the Baseline, no recovery training). We report two speedup measures: *Act. Speedup*, the per-action inference speedup reflecting how fast the model generates a single action; and *Task Speedup* = Act. Speedup / Step Ratio, the end-to-end speedup for completing all evaluation episodes.

DTR-16 achieves **1.64**× task speedup and also reduces memory by 42%. Zero-shot attention dropping yields marginal task speedup (Attn Drop 8: 1.09×) due to limited per-step gains. Critically, zero-shot block dropping is *slower* end-to-end despite faster per-action inference: Block Drop 4 achieves 1.05× action speedup but only 0.72× task speedup because severe SR degradation (78%) inflates total steps. This shows that recovery training is not merely beneficial but necessary, without it, the step overhead from degraded SR more than offsets the per-action speedup. For more details, please refer to Appendix K.

6.2 Cross-benchmark analysis: what benchmarks do we need?

Language-block recoverability holds across benchmarks but shrinks under distribution shift: the removed capacity acts as a robustness reserve beyond language understanding.

Following the compute-matched setting from Section 6.1, we evaluate DTR on LIBERO-Plus (Fei et al., 2025) (Table 6) and RoboTwin 2.0 (Chen et al., 2025a) (Figure 4; full results in Table 17).

Language backbone redundancy is consistent across benchmarks, but the magnitude varies. At Drop-9, LIBERO *exceeds* the baseline (92.3% vs. 91.7%). On LIBERO-Plus, performance drops by 3.8%. On RoboTwin 2.0, Easy variants decrease by only 0.6%, while Hard variants degrade by 6.6%. Furthermore, we make two key observations: (i) on LIBERO-Plus, the largest degradation after dropping is not in the Language category (-5.1 at Drop-9) but in Robot (-10.6), which perturbs the arm's initial pose. (ii) RoboTwin Hard variants degrade much more sharply than Easy variants across all tasks (Figure 4). These suggest that the language backbone, despite being over-parameterized for simple instructions, contributes to the model's *generalization* under distribution shift. The practical impact of compression thus depends on deployment robustness requirements. *Consequently, we need*

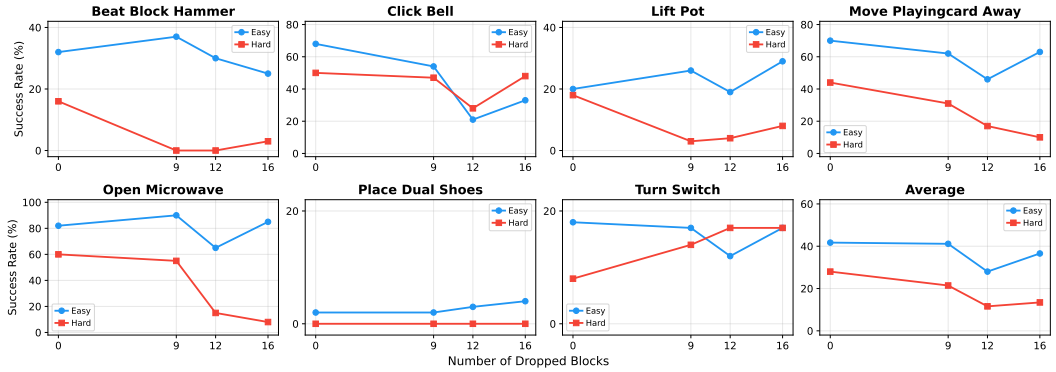


Figure 4: Per-task DTR results on RoboTwin 2.0 with $\pi_{0.5}$.

Table 6: Per-perturbation-category results on $\pi_{0.5}$ / LIBERO-Plus (bsz 32 for 30K steps). Subscripts indicate relative % change from baseline. Darker red indicates larger degradation.

Setting	Size	FLOPs	Camera	Robot	Language	Light	Background	Noise	Layout	Avg.
Baseline	100%	100%	85.4	60.3	70.9	91.5	92.5	87.0	82.1	81.4
Drop-9	60.6%	57.9%	85.4$_{-0.0}$	49.7$_{-10.6}$	65.8$_{-5.1}$	89.7$_{-1.8}$	89.8$_{-2.7}$	85.2$_{-1.8}$	77.6$_{-4.5}$	77.6$_{-3.8}$
Drop-12	47.5%	43.8%	81.8$_{-3.6}$	43.0$_{-17.3}$	59.7$_{-11.2}$	86.9$_{-4.6}$	87.6$_{-4.9}$	78.3$_{-8.7}$	73.6$_{-8.5}$	73.0$_{-8.4}$
Drop-16	30.0%	25.1%	72.7$_{-12.7}$	36.1$_{-24.2}$	61.0$_{-9.9}$	86.8$_{-4.7}$	82.8$_{-9.7}$	72.0$_{-15.0}$	70.4$_{-11.7}$	68.8$_{-12.6}$
Drop-17	25.6%	20.4%	73.7$_{-11.7}$	32.1$_{-28.2}$	62.3$_{-8.6}$	83.3$_{-8.2}$	79.5$_{-13.0}$	73.0$_{-14.0}$	71.8$_{-10.3}$	68.0$_{-13.4}$

benchmarks with higher language complexity and sufficiently large out-of-domain perturbations to match the substantial language components in VLA models.

6.3 Cross-model analysis: how should future VLAs be designed?

Language redundancy persists across architectures, rooted in the VLM-to-VLA capacity mismatch.

Table 7 follows the setting of Table 1 and applies the same Drop Half Language protocol to four VLA architectures. Under the same batch size and number of training steps, all four models match or exceed their baselines after removing half of the LLM blocks, with OpenVLA-OFT rebounding by +3.3 points. This is partly due to the saturation of LIBERO, and it is consistent with a structural mismatch rather than a scale effect: VLA models inherit language capacity far beyond what short robotic instructions require. *Therefore, future VLA architecture designs can appropriately reduce the language component to better match task demands.*

Table 7: **DTR** across VLA architectures.

Model	Drop	Spatial	Object	Goal	Long	Avg.
OpenVLA-OFT	0/32	97.2	98.4	95.6	88.6	95.0
	16/32	**99.0**	**100.0**	**97.8**	**96.4**	**98.3**
$\pi_{0.5}$	0/18	96.6	95.0	93.0	82.0	91.7
	9/18	**98.8**	**98.6**	**93.4**	**82.4**	**93.3**
GigaBrain-0	0/26	84.4	98.2	93.0	**76.2**	**88.0**
	13/26	**85.0**	**98.6**	**93.2**	75.2	**88.0**
Lingbot-VLA	0/36	81.8	95.0	**86.6**	**67.8**	82.8
	18/36	**85.6**	**97.0**	84.2	**67.8**	**83.7**

7 Conclusion

We presented **DTR** and **`GateProbe`** to systematically probe architectural redundancy in VLA models. Our findings reveal a striking asymmetry: on standard manipulation benchmarks, language backbones inherited from pretrained VLMs carry far more capacity than the evaluated tasks demand, while vision and action pathways remain critical and compress poorly. Both simulation and real-world experiments confirm this pattern across diverse VLA architectures. These results point to a fundamental mismatch between the capacity distribution of today's VLA models and the actual computational demands of closed-loop control, calling for future architectures that allocate capacity more deliberately and future benchmarks that place stronger pressure on compositional language grounding and OOD generalization.

AI USE STATEMENT

We used generative AI tools for three tasks in this work. A large language model was used to edit the manuscript for grammar and readability, and the authors reviewed each suggested edit before keeping it. AI assistance was used to operate the experiments, that is, to launch training and evaluation jobs across machines and to monitor their progress. It was also used to assemble the per-run outputs into the tables and figures reported here; every number obtained this way was checked by hand against the raw evaluation logs before we reported it.

REPRODUCIBILITY STATEMENT

We describe the DTR protocol and **`GateProbe`** in Section 3, with full derivations and implementation details of **`GateProbe`** in Appendix D. All models, benchmarks and evaluation settings are documented in Appendix A, the exact dropped-block index sets in Appendices F and G, and all training hyperparameters in Appendix H. Per-task results supporting the aggregate numbers are reported in Appendices I–K. The code will be released upon acceptance of the paper.

REFERENCES

Hongxiao Bai and Yun Li. Structured sparsity in the NVIDIA ampere architecture and applications in search engines. NVIDIA Technical Blog, 2023. URL `https://developer.nvidia.com/blog/structured-sparsity-in-the-nvidia-ampere-architecture-and-applications-in-search-engines/`. Accessed May 6, 2026.

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

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems*, 2025b. doi: 10.15607/RSS.2025.XXI.010.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, Weiliang Deng, Yubin Guo, Tian Nian, Xuanbing Xie, Qiangyu Chen, Kailun Su, Tianling Xu, Guodong Liu, Mengkang Hu, Huan-ang Gao, Kaixuan Wang, Zhixuan Liang, Yusen Qin, Xiaokang Yang, Ping Luo, and Yao Mu. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025a.

Xinrui Chen, Hongxing Zhang, Fanyi Zeng, Yongxian Wei, Yizhi Wang, Xitong Ling, Guanghao Li, and Chun Yuan. Prune&comp: Free lunch for layer-pruned llms via iterative pruning with magnitude compensation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 20316–20324, 2026.

Yuxuan Chen, Yixin Han, Yize Huang, and Xiao Li. RLRC: Reinforcement learning-based recovery for compressed vision-language-action models. *arXiv preprint arXiv:2506.17639*, 2025b.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. LIBERO-Plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10323–10337. PMLR, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023.

Gemini Robotics Team et al. Gemini robotics: Bringing AI into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

GigaBrain Team, Angen Ye, Boyuan Wang, Chaojun Ni, Guan Huang, Guosheng Zhao, Haoyun Li, Jie Li, Jiagang Zhu, Lv Feng, Peng Li, Qiuping Deng, Runqi Ouyang, Wenkang Qin, Xinze Chen, Xiaofeng Wang, Yang Wang, Yifan Li, Yilong Li, Yiran Ding, Yuan Xu, Yun Ye, Yukun Zhou, Zhehao Dong, Zhenan Wang, Zhichao Liu, and Zheng Zhu. GigaBrain-0: A world model-powered vision-language-action model. *arXiv preprint arXiv:2510.19430*, 2025.

Bryce Grant, Xijia Zhao, and Peng Wang. Not all features are created equal: A mechanistic study of vision-language-action models. *arXiv preprint arXiv:2603.19233*, 2026.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A. Roberts. The unreasonable ineffectiveness of the deeper layers. In *International Conference on Learning Representations*, 2025.

Shwai He, Guoheng Sun, Zheyu Shen, and Ang Li. What matters in transformers? not all attention is needed. *arXiv preprint arXiv:2406.15786*, 2024.

Wei Huang, Anda Cheng, and Yinggui Wang. Gradpruner: Gradient-guided layer pruning enabling efficient fine-tuning and inference for llms. *arXiv preprint arXiv:2601.19503*, 2026.

Jason Jabbour, Dong-Ki Kim, Max Smith, Jay Patrikar, Radhika Ghosal, Youhui Wang, Ali Agha, Vijay Janapa Reddi, and Shayegan Omidshafiei. Don't run with scissors: Pruning breaks VLA models but they can be recovered. *arXiv preprint arXiv:2510.08464*, 2025.

Boseong Jeon, Yunho Choi, and Taehan Kim. Shallow-π: Knowledge distillation for flow-based VLAs. *arXiv preprint arXiv:2601.20262*, 2026.

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

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025b.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791, 2023. Datasets and Benchmarks Track.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. ShortGPT: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204, 2025. doi: 10.18653/v1/2025.findings-acl.1035.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017.

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11264–11272, 2019.

NVIDIA. NVIDIA Jetson Thor: Advanced AI for physical robotics. NVIDIA product page, 2026a. URL https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-thor/. Accessed May 6, 2026.

NVIDIA. Working with quantized types. NVIDIA TensorRT Documentation, 2026b. URL https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/work-quantized-types.html. Accessed May 6, 2026.

NVIDIA. Data formats and tensors. NVIDIA TensorRT Documentation, 2026c. URL https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/data-formats-tensors.html. Accessed September 14, 2026.

Open X-Embodiment Collaboration, Abby O'Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, et al. Open x-embodiment: Robotic learning datasets and RT-X models. *arXiv preprint arXiv:2310.08864*, 2023.

Jiwon Song, Kyungseok Oh, Taesu Kim, Hyungjun Kim, Yulhwa Kim, and Jae-Joon Kim. SLEB: Streamlining LLMs through redundancy verification and elimination of transformer blocks. *arXiv preprint arXiv:2402.09025*, 2024.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Hanzhen Wang, Jiaming Xu, Yushun Xiang, Jiayi Pan, Yongkang Zhou, Yong-Lu Li, and Guohao Dai. SpecPrune-VLA: Accelerating vision-language-action models via action-aware self-speculative pruning. *arXiv preprint arXiv:2509.05614*, 2025a.

Hongyu Wang, Chuyan Xiong, Ruiping Wang, and Xilin Chen. BitVLA: 1-bit vision-language-action models for robotics manipulation. *arXiv preprint arXiv:2506.07530*, 2025b.

Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Qian Zhu, He Sun, Yong Wang, Shuailei Ma, Yiyu Ren, Kejia Zhang, Hui Yu, Jingmei Zhao, Shuai Zhou, Zhenqi Qiu, Houlong Xiong, Ziyu Wang, Zechen Wang, Ran Cheng, Yong-Lu Li, Yongtao Huang, Xing Zhu, Yujun Shen, and Kecheng Zheng. A pragmatic VLA foundation model. *arXiv preprint arXiv:2601.18692*, 2026.

Yuhao Xu, Yantai Yang, Zhenyang Fan, Yufan Liu, Yuming Li, Bing Li, and Zhipeng Zhang. QVLA: Not all channels are equal in vision-language-action model's quantization. *arXiv preprint arXiv:2602.03782*, 2026.

Yantai Yang, Yuhao Wang, Zichen Wen, Zhongwei Luo, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. EfficientVLA: Training-free acceleration and compression for vision-language-action models. *arXiv preprint arXiv:2506.10100*, 2025.

Rongyu Zhang, Menghang Dong, Yuan Zhang, Liang Heng, Xiaowei Chi, Gaole Dai, Li Du, Yuan Du, and Shanghang Zhang. MoLe-VLA: Dynamic layer-skipping vision language action model via mixture-of-layers for efficient robot manipulation. *arXiv preprint arXiv:2503.20384*, 2025.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong T. Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023.

A MODEL AND BENCHMARK DETAILS

Models. We evaluate **DTR** on four VLA architectures spanning different backbones, scales, and action-head designs. (i) $\pi_{0.5}$ (Black et al., 2025a): a dual-stream flow-matching architecture with a PaliGemma language backbone (18 Gemma layers) and a separate Gemma action expert (18 layers), using SigLIP as the vision encoder. (ii) **OpenVLA-OFT** (Kim et al., 2025a): fine-tunes OpenVLA with a Llama-2-7B backbone (32 layers), SigLIP vision encoder, and an MLP action head. (iii) **Lingbot-VLA** (Wu et al., 2026): built on Qwen2.5-VL-3B (36 layers) with a Qwen2 action expert (36 layers) and a flow-matching action decoder, totaling approximately 4B parameters. (iv) **GigaBrain-0** (GigaBrain Team et al., 2025): uses PaliGemma2-3B with integrated Gemma2 expert layers (26 layers), SigLIP vision encoder, and a diffusion-based action head, totaling approximately 3.5B parameters.

Benchmarks. We use three simulation benchmarks. **LIBERO** (Liu et al., 2023) contains four task suites: Spatial, Object, Goal, and Long/Libero-10, with 10 tasks per suite. We evaluate each task over 50 trials. **LIBERO-Plus** (Fei et al., 2025) extends LIBERO with increased visual and physical diversity, including perturbations in background, texture, viewpoint, robot pose, language, lighting, noise, and layout. **RoboTwin 2.0** (Chen et al., 2025a) targets dual-arm manipulation and is used to evaluate whether language-block redundancy transfers to more complex manipulation settings.

B BLOCK DROPPING IN JOINT-ATTENTION ARCHITECTURES

In $\pi_{0.5}$ (Black et al., 2025a), the language backbone (PaliGemma) and action expert (Gemma) are parallel transformer streams with identical layer counts. At each layer, both streams perform *joint attention*: the language and action tokens are concatenated along the sequence dimension, and a shared attention matrix is computed over the combined sequence. This means the action expert's queries attend over both its own keys/values and those from the language backbone.

This architecture has a direct implication for block dropping. When we drop a language block, we cannot remove all of its parameters:

(i) **Retained parameters.** The language block's *key projection* (W_K), *value projection* (W_V), and *input layer normalization* must remain active, because the action expert still needs to cross-attend to language representations at this layer.

(ii) **Removed parameters.** The language block's *query projection* (W_Q), *output projection* (W_O), and the entire *MLP sublayer* are eliminated. The language hidden state passes through as identity: $h_i^{\mathcal{L}} = h_{i-1}^{\mathcal{L}}$.

(iii) **Gradient flow.** Even after dropping, the retained K/V projections still receive gradients from the action expert's loss during recovery fine-tuning, effectively repurposing them as cross-attention adapters.

As a result, dropping a language block in $\pi_{0.5}$ removes approximately 75% of that block's parameters (Q, O, and MLP) rather than 100%. We account for this when computing compression ratios throughout the paper. For architectures without joint attention (*e.g.*, OpenVLA-OFT, where the action head is a separate MLP), dropping a language block removes all of its parameters.

C ACTION HEAD COMPRESSION IN OPENVLA-OFT

OpenVLA-OFT uses an MLPResNet as its action head, consisting of an input projection ($4096 \to d_h$), two residual MLP blocks ($d_h \to d_h$), and an output projection ($d_h \to 7$), where d_h is the hidden dimension. Unlike the language backbone (which uses block dropping), we compress the action head by reducing d_h: (i) **Drop Half**: $d_h = 4096 \to 2048$, reducing action head parameters from 50.4M to 16.8M (model-wide: 99.5% of original size). (ii) **Extreme**: $d_h = 4096 \to 256$, reducing to 1.3M (model-wide: 99.3%). Since the action head accounts for only $\sim$0.7% of OpenVLA-OFT's total parameters, even aggressive action compression has negligible impact on model size, but can significantly affect task performance (Section 4.2).

D GATEPROBE DETAILS

Taylor-expansion interpretation. $I_{\text{gate}}(B_i)$ is the first-order Taylor approximation of the loss change when the block's contribution is scaled to zero:

$$\mathcal{L}|_{\alpha_i=0} \approx \mathcal{L}|_{\alpha_i=1} - 1 \cdot \frac{\partial \mathcal{L}}{\partial \alpha_i}\bigg|_{\alpha_i=1}. \tag{7}$$

A larger $|I_{\text{gate}}(B_i)|$ means removing the block causes a larger immediate loss increase, suggesting it is more important and should be retained.

Implementation. The gate in **GateProbe** is virtual: we never modify the model architecture or introduce trainable parameters. We register forward pre-hooks at each block's input normalization layer to capture hidden states h_{i-1} and h_i, call `retain_grad()` on these tensors, and run a standard forward-backward pass. The gate score is then computed as the inner product $\langle \partial \mathcal{L}/\partial h_i, F_i(h_{i-1}) \rangle$ (Eq. 6 in the main text). This requires only a single forward-backward pass over a small calibration set.

Algorithm 1: **GateProbe** Importance Profiling

Input: Model $\mathcal{M}$ with N blocks, calibration data $\{(o_j, p_j, a_j)\}_{j=1}^{M}$, task loss $\mathcal{L}$
Output: Importance scores $\{I_{\text{gate}}(B_i)\}_{i=1}^{N}$

1. Initialize accumulators: $S_i \leftarrow 0$ for $i = 1, \ldots, N$
2. **for** each calibration batch (o, p, a) **do**
3. Register forward pre-hooks on each block's input norm to capture h_i and call `retain_grad()`
4. Forward pass: compute $\mathcal{L}(o, p, a)$
5. Backward pass: compute $\partial \mathcal{L}/\partial h_i$ for all i
6. **for** $i = 1, \ldots, N$ **do**
7. $F_i \leftarrow h_i - h_{i-1}$ ▷ block residual
8. $g_i \leftarrow \partial \mathcal{L}/\partial h_i$ ▷ downstream gradient
9. $S_i \leftarrow S_i + |\langle g_i,\, F_i \rangle|$ ▷ gate sensitivity
10. Remove hooks, free intermediate states
11. $I_{\text{gate}}(B_i) \leftarrow S_i/M$ for all i
12. **return** $\{I_{\text{gate}}(B_i)\}_{i=1}^{N}$, sorted in descending order

E BLOCK IMPORTANCE METRICS

Table 3 compares eight importance metrics for selecting which blocks to drop, most of which originate from prior work on LLM layer pruning. We classify them along two axes: whether they require gradient computation, and whether they operate in parameter space or activation space.

Gradient-based, activation-space.

GateProbe (ours) places a virtual scalar gate α_l (initialized to 1) on the residual branch of each block and measures the sensitivity of the task loss to this gate:

$$S_l = \mathbb{E}\left[\left|\left\langle \frac{\partial \mathcal{L}}{\partial \mathbf{h}_l},\ F_l(\mathbf{h}_{l-1}) \right\rangle\right|\right], \tag{8}$$

where $F_l(\mathbf{h}_{l-1}) = \mathbf{h}_l - \mathbf{h}_{l-1}$ is the block's residual output and $\mathbf{h}_l$ is the hidden state after block l. This is the first-order Taylor approximation of the loss change when scaling the residual by α_l; GateProbe thus belongs to the first-order Taylor family (Molchanov et al., 2017), but differs from parameter-space variants in three respects: it gates each block's *residual contribution*, matching block removal ($h_i = h_{i-1}$) in pre-LN architectures; it aggregates $\mathbb{E}[|\cdot|]$ over samples to avoid sign cancellation; and it is supervised by the downstream action loss and validated by post-recovery success rate. No model modification is needed: hidden states and gradients are captured via hooks. Cost: one forward + backward pass over calibration data.

Gradient-based, parameter-space.

Taylor (Ma et al., 2023; Chen et al., 2026) accumulates signed first-order salience across calibration batches, then takes the absolute value:

$$S_l = \frac{1}{N} \sum_{w \in l} \left| \sum_{t=1}^{N} \frac{\partial \mathcal{L}_t}{\partial w} \cdot w \right|. \tag{9}$$

Signed accumulation allows positive and negative contributions to cancel, reducing noise. Cost: one forward + backward pass (same as IGIA, but accumulation differs).

IGIA (Huang et al., 2026) (diagonal Fisher approximation) sums squared gradients per layer across all batches:

$$S_l = \sum_{t=1}^{N} \sum_{w \in l} \left(\frac{\partial \mathcal{L}_t}{\partial w} \right)^2. \tag{10}$$

This approximates the diagonal of the Fisher Information Matrix. Cost: one forward + backward pass.

Fisher (Molchanov et al., 2019; Chen et al., 2026) (OBD-style) weights squared gradients by squared parameter magnitudes:

$$S_l = \frac{1}{N} \sum_{t=1}^{N} \sum_{w \in l} \left(\frac{\partial \mathcal{L}_t}{\partial w} \right)^2 w^2. \tag{11}$$

This estimates the loss change from removing each weight under a diagonal Hessian approximation. Cost: one forward + backward pass.

Hessian trace (Chen et al., 2026) uses Hutchinson's stochastic estimator with Hessian-vector products:

$$S_l = |\mathrm{Tr}(\mathbf{H}_l)| \approx \left| \frac{1}{K} \sum_{k=1}^{K} \mathbf{z}_k^\top \mathbf{H}_l \mathbf{z}_k \right|, \quad \mathbf{z}_k \sim \text{Rademacher}, \tag{12}$$

where $\mathbf{H}_l = \partial^2 \mathcal{L} / \partial \mathbf{w}_l^2$. The Hessian-vector product is computed via double backpropagation without forming $\mathbf{H}_l$ explicitly. Cost: one forward + two backward passes (requires `create_graph=True`).

Gradient-free, activation-space.

CosSim (Block Influence) measures how much each block transforms its input:

$$S_l = 1 - \mathbb{E}[\cos(\mathbf{h}_{l-1},\, \mathbf{h}_l)]\,. \tag{13}$$

Blocks with high cosine similarity (low S_l) are near-identity and considered more droppable. Proposed by ShortGPT (Men et al., 2025) and widely adopted as a standard baseline (He et al., 2024; Chen et al., 2026). Cost: one forward pass.

CosSim (contig.) extends this to contiguous block groups. For each possible starting position s, it computes the angular distance between $\mathbf{h}_s$ and $\mathbf{h}_{s+n}$ and selects the contiguous window with smallest distance. Follows Gromov et al. (2025). Cost: one forward pass.

PPL (Chen et al., 2026) (leave-one-out) drops each block individually and measures the resulting loss increase:

$$S_l = \mathcal{L}(\text{model w/o block } l) - \mathcal{L}(\text{baseline}). \tag{14}$$

Cost: $L + 1$ forward passes (L = number of layers), making it the most expensive non-gradient method.

Gradient-free, parameter-space. **Magnitude** (Chen et al., 2026) sums the L1 norm of all parameters in each layer:

$$S_l = \sum_{w \in l} |w|. \tag{15}$$

This is data-independent and deterministic. Cost: zero (no forward pass needed).

Summary. Table 8 summarizes the classification, computational cost, and wall-clock time measured on $\pi_{0.5}$ (18 PaliGemma blocks) with 64 calibration batches of size 8 on a single H200 GPU.

Table 8: Classification of block importance metrics. Wall-clock time measured on $\pi_{0.5}$ with 64 calibration batches (batch size 8) on one H200 GPU, excluding model and data loading.

Metric	Gradient	Space	Cost	Time (s)
GateProbe (ours)	Yes	Activation	1 fwd + 1 bwd	24.9
IGIA	Yes	Parameter	1 fwd + 1 bwd	25.4
Fisher	Yes	Parameter	1 fwd + 1 bwd	26.1
Hessian trace	Yes	Parameter	1 fwd + 2 bwd	77.2
Taylor	Yes	Parameter	1 fwd + 1 bwd*	471.7
CosSim	No	Activation	1 fwd	9.4
CosSim (contig.)	No	Activation	1 fwd	9.2
PPL	No	Activation	$(L+1)$ fwd	160.8
Magnitude	No	Parameter	0	0.2

* Taylor requires per-parameter signed accumulation on CPU across batches, causing significant memory-transfer overhead despite the same forward/backward count as IGIA and Fisher.

F DROP INDEX LOOKUP TABLES

Tables 9, 10, and 11 list the kept PaliGemma block indices for all drop configurations used in this paper. Table 9 covers the metric comparison on LIBERO (Table 3); metrics sharing the same row (marked †) select identical blocks at that drop level. Tables 10 and 11 cover the **GateProbe** selections for LIBERO-Plus (Table 6) and RoboTwin 2.0 (Table 17), which use dataset-specific profiling and therefore differ from the LIBERO selections.

Table 9: Kept PaliGemma block indices for each metric and drop configuration on $\pi_{0.5}$ / LIBERO (18 blocks, indexed 0–17). Blocks not listed are dropped.

Setting	Metric	Keep Blocks
Drop-9 (9/18)	**GateProbe**	[0,1,2,3,4,5,6,8,9]
	Taylor / IGIA	[0,1,3,4,5,6,7,8,9]
	Fisher	[0,1,2,3,4,5,6,7,8]
	Hessian	[0,1,2,3,5,8,9,11,13]
	PPL	[0,1,4,7,8,10,11,13,14]
	CosSim (contig.)	[0,1,11,12,13,14,15,16,17]
	Magnitude	[2,3,6,12,13,14,15,16,17]
	CosSim	[0,1,2,12,13,14,15,16,17]
Drop-12 (12/18)	**GateProbe**	[0,1,2,5,6,8]
	Taylor	[0,3,4,5,6,8]
	IGIA	[0,1,3,4,6,7]
	Fisher	[0,1,5,6,7,8]
	Hessian	[0,1,3,8,9,13]
	PPL	[0,1,4,7,10,13]
	CosSim	[0,1,14,15,16,17]
	Magnitude	[2,13,14,15,16,17]
	CosSim (contig.)	[0,13,14,15,16,17]
Drop-16 (16/18)	**GateProbe**/ Fisher	[0,5]
	Taylor	[5,6]
	IGIA	[0,7]
	Hessian	[0,1]
	PPL	[0,13]
	Mag. / CosSim / CosSim (c.)	[16,17]
Drop-17 (17/18)	**GateProbe**/ Fisher / Hessian / IGIA / PPL	[0]
	Taylor	[5]
	Mag. / CosSim / CosSim (c.)	[17]

Table 10: Kept PaliGemma block indices for **GateProbe** on $\pi_{0.5}$ / LIBERO-Plus (18 blocks, indexed 0–17).

Setting	Keep Blocks
Drop-9 (9/18)	[0,1,3,4,5,6,7,8,9]
Drop-12 (12/18)	[0,1,4,5,6,7]
Drop-16 (16/18)	[0,5]
Drop-17 (17/18)	[0]

Table 11: Kept PaliGemma block indices for **GateProbe** on $\pi_{0.5}$ / RoboTwin 2.0 (18 blocks, indexed 0–17). Each task uses its own per-task **GateProbe** profiling.

Task	Drop-9	Drop-12	Drop-16
Beat Block Hammer	[0,1,2,3,4,5,6,8,13]	[0,1,2,3,5,6]	[0,1]
Click Bell	[0,1,2,3,4,5,6,11,12]	[0,1,2,3,5,11]	[0,2]
Lift Pot	[0,1,2,3,4,5,6,11,12]	[0,1,2,3,4,5]	[0,5]
Move Playingcard	[0,1,2,3,4,5,6,7,9]	[0,1,2,3,5,6]	[0,2]
Open Microwave	[0,1,2,3,4,5,6,7,13]	[0,1,2,5,6,7]	[0,1]
Place Dual Shoes	[0,1,2,3,4,5,6,7,13]	[0,1,2,3,4,5]	[0,1]
Turn Switch	[0,1,2,3,4,5,6,9,13]	[0,1,2,3,5,6]	[0,2]

G VISION DROP LISTS

For all vision-dropping experiments, the attention and MLP drop lists are identical. We report the retained vision blocks below. All layer indices are zero-based.

Table 12: Retained vision block indices for $\pi_{0.5}$ vision-drop experiments. The vision backbone is a SigLIP tower with 27 layers, indexed 0–26.

Setting	Retained Blocks
Drop Half	$[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26]$
Keep 2	$[0, 26]$

For OpenVLA-OFT, the DINO tower follows the OpenVLA feature-extraction path, where the final visual representation is taken from the second-to-last effective block. Therefore, in the Keep 2 setting, layer 23 is retained as the DINO endpoint, while layer 25 is retained as the corresponding SigLIP endpoint.

H TRAINING DETAILS

We use the same recovery fine-tuning protocol as the corresponding full-model baseline whenever possible. This ensures that performance differences mainly reflect the effect of dropping rather than changes in data, optimization, or training budget. For compute-matched comparisons, we scale the training budget according to the compute reduction of the dropped model while keeping the data and optimization protocol unchanged.

LIBERO. We train and recover all models on the mixed LIBERO training set, combining the Spatial, Object, Goal, and Long suites from the modified LIBERO RLDS datasets. The main training settings are summarized in Table 14.

For OpenVLA-OFT, we follow the original fine-tuning recipe with LoRA adaptation, L1 action regression, two input images, proprioceptive inputs, and image augmentation. For $\pi_{0.5}$, we use the standard LIBERO training configuration with a 30K-step budget, global batch size 32, bfloat16 precision, and a learning rate of 5×10^{-5} after a 10K-step warmup.

Table 13: Retained vision block indices for OpenVLA-OFT vision-drop experiments. OpenVLA-OFT uses a fused DINO+SigLIP vision backbone: DINO has 24 layers indexed 0–23, and SigLIP has 27 layers indexed 0–26.

Setting	**DINO Retained Blocks**	**SigLIP Retained Blocks**
Drop Half	$[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22]$	$[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26]$
Keep 2	$[0, 23]$	$[0, 25]$

Table 14: Training settings on LIBERO.

Setting	OpenVLA-OFT	$\pi_{0.5}$
Base checkpoint	OpenVLA-7B	$\pi_{0.5}$ base checkpoint
Training data	Spatial, Object, Goal, Long	Spatial, Object, Goal, Long
Objective	L1 action regression	Flow-matching action prediction
Optimizer	AdamW	AdamW
Global batch size	16	32
Training steps	50K	30K
Learning rate	5×10^{-4}	5×10^{-5}
LR schedule	decay after 30K steps	10K warmup, then constant LR
Precision	bfloat16	bfloat16
Fine-tuning	LoRA, rank 32, dropout 0.0	full fine-tuning
Inputs	2 images + proprioception	LIBERO observation + robot state
Image augmentation	enabled	default setting

RoboTwin 2.0. We train and recover all models on the mixed 7 RoboTwin 2.0 training set, combining trajectories from the selected RoboTwin 2.0 manipulation tasks. The main training settings are summarized in Table 15.

Table 15: Training settings on RoboTwin 2.0.

Setting	OpenVLA-OFT	$\pi_{0.5}$
Base checkpoint	OpenVLA-7B	$\pi_{0.5}$ base checkpoint
Training data	mixed 7 RoboTwin 2.0 tasks	mixed 7 RoboTwin 2.0 tasks
Objective	L1 action regression	Flow-matching action prediction
Optimizer	AdamW	AdamW
Global batch size	16	32
Training steps	100K	30K
Learning rate	5×10^{-4}	5×10^{-5}
LR schedule	decay after 30K steps	10K warmup, then constant LR
Precision	bfloat16	bfloat16
Fine-tuning	LoRA, rank 32, dropout 0.0	full fine-tuning
Inputs	3 images + proprioception	3 camera views + robot state
Image augmentation	enabled	default setting

For OpenVLA-OFT, we use LoRA adaptation with rank 32, L1 action regression, three input images, proprioceptive inputs, and image augmentation. For $\pi_{0.5}$, we use the RoboTwin 2.0 mixed-task training configuration with a 30K-step budget, global batch size 32, bfloat16 precision, and a learning rate of 5×10^{-5} after a 10K-step warmup.

I PER-TASK RESULTS ON LIBERO-PLUS

Table 16 provides the full per-task results on LIBERO-Plus.

Table 16: Per-perturbation-category breakdown on LIBERO-Plus for each task suite with **`GateProbe`** block selection. Subscripts indicate absolute difference from baseline.

Setting	Size	FLOPs	Camera	Robot	Language	Light	Background	Noise	Layout	Avg.
Spatial										
Baseline	100%	100%	85.4	64.6	72.1	89.0	92.2	87.5	93.8	83.0
Drop-9	60.6%	57.9%	85.9$_{+0.5}$	63.7$_{-0.9}$	74.6$_{+2.5}$	87.3$_{-1.7}$	91.5$_{-0.7}$	86.0$_{-1.5}$	90.4$_{-3.4}$	82.4$_{-0.6}$
Drop-12	47.5%	43.8%	83.2$_{-2.2}$	50.6$_{-14.0}$	70.3$_{-1.8}$	86.3$_{-2.7}$	89.1$_{-3.1}$	80.6$_{-6.9}$	87.3$_{-6.5}$	77.6$_{-5.4}$
Drop-16	30.0%	25.1%	81.1$_{-4.3}$	53.7$_{-10.9}$	72.3$_{+0.2}$	90.4$_{+1.4}$	91.5$_{-0.7}$	82.3$_{-5.2}$	85.5$_{-8.3}$	78.8$_{-4.2}$
Drop-17	25.6%	20.4%	86.7$_{+1.3}$	46.6$_{-18.0}$	74.1$_{+2.0}$	91.8$_{+2.8}$	94.6$_{+2.4}$	87.7$_{+0.2}$	90.1$_{-3.7}$	81.0$_{-2.0}$
Object										
Baseline	100%	100%	95.7	56.5	83.6	99.0	98.8	96.2	92.3	88.1
Drop-9	60.6%	57.9%	96.5$_{+0.8}$	34.2$_{-22.3}$	76.0$_{-7.6}$	98.7$_{-0.3}$	98.4$_{-0.4}$	97.2$_{+1.0}$	89.1$_{-3.2}$	83.1$_{-5.0}$
Drop-12	47.5%	43.8%	95.2$_{-0.5}$	32.2$_{-24.3}$	76.8$_{-6.8}$	97.0$_{-2.0}$	95.6$_{-3.2}$	92.2$_{-4.0}$	84.4$_{-7.9}$	80.7$_{-7.4}$
Drop-16	30.0%	25.1%	91.9$_{-3.8}$	29.9$_{-26.6}$	79.4$_{-4.2}$	97.6$_{-1.4}$	93.5$_{-5.3}$	96.2$_{0.0}$	84.4$_{-7.9}$	80.7$_{-7.4}$
Drop-17	25.6%	20.4%	93.2$_{-2.5}$	27.6$_{-28.9}$	81.6$_{-2.0}$	94.3$_{-4.7}$	89.9$_{-8.9}$	94.1$_{-2.1}$	82.9$_{-9.4}$	79.5$_{-8.6}$
Goal										
Baseline	100%	100%	79.9	61.6	62.9	94.3	93.6	83.1	61.4	74.8
Drop-9	60.6%	57.9%	88.2$_{+8.3}$	57.2$_{-4.4}$	53.4$_{-9.5}$	93.5$_{-0.8}$	95.0$_{+1.4}$	85.8$_{+2.7}$	61.6$_{+0.2}$	74.4$_{-0.4}$
Drop-12	47.5%	43.8%	79.4$_{-0.5}$	45.7$_{-15.9}$	44.1$_{-18.8}$	88.2$_{-6.1}$	88.6$_{-5.0}$	75.5$_{-7.6}$	57.4$_{-4.0}$	66.3$_{-8.5}$
Drop-16	30.0%	25.1%	48.3$_{-31.6}$	19.8$_{-41.8}$	30.7$_{-32.2}$	84.9$_{-9.4}$	74.0$_{-19.6}$	42.7$_{-40.4}$	49.9$_{-11.5}$	47.2$_{-27.6}$
Drop-17	25.6%	20.4%	56.6$_{-23.3}$	20.5$_{-41.1}$	39.3$_{-23.6}$	79.2$_{-15.1}$	71.9$_{-21.7}$	53.3$_{-29.8}$	55.8$_{-5.6}$	51.6$_{-23.2}$
Long										
Baseline	100%	100%	80.9	58.8	66.3	83.2	86.2	81.3	82.7	76.4
Drop-9	60.6%	57.9%	71.8$_{-9.1}$	45.0$_{-13.8}$	60.6$_{-5.7}$	78.5$_{-4.7}$	75.8$_{-10.4}$	72.8$_{-8.5}$	68.6$_{-14.1}$	66.9$_{-9.5}$
Drop-12	47.5%	43.8%	70.2$_{-10.7}$	44.5$_{-14.3}$	49.6$_{-16.7}$	75.2$_{-8.0}$	78.5$_{-7.7}$	65.9$_{-15.4}$	64.7$_{-18.0}$	63.1$_{-13.3}$
Drop-16	30.0%	25.1%	70.6$_{-10.3}$	43.8$_{-15.0}$	64.8$_{-1.5}$	73.0$_{-10.2}$	74.4$_{-11.8}$	65.7$_{-15.6}$	61.5$_{-21.2}$	64.2$_{-12.2}$
Drop-17	25.6%	20.4%	60.4$_{-20.5}$	35.9$_{-22.9}$	56.9$_{-9.4}$	66.4$_{-16.8}$	64.4$_{-21.8}$	58.1$_{-23.2}$	56.7$_{-26.0}$	56.3$_{-20.1}$
Average (all 4 suites)										
Baseline	100%	100%	85.4	60.3	70.9	91.5	92.5	87.0	82.1	81.4
Drop-9	60.6%	57.9%	85.4$_{0.0}$	49.7$_{-10.6}$	65.8$_{-5.1}$	89.7$_{-1.8}$	89.8$_{-2.7}$	85.2$_{-1.8}$	77.6$_{-4.5}$	77.6$_{-3.8}$
Drop-12	47.5%	43.8%	81.8$_{-3.6}$	43.0$_{-17.3}$	59.7$_{-11.2}$	86.9$_{-4.6}$	87.6$_{-4.9}$	78.3$_{-8.7}$	73.6$_{-8.5}$	73.0$_{-8.4}$
Drop-16	30.0%	25.1%	72.7$_{-12.7}$	36.1$_{-24.2}$	61.0$_{-9.9}$	86.8$_{-4.7}$	82.8$_{-9.7}$	72.0$_{-15.0}$	70.4$_{-11.7}$	68.8$_{-12.6}$
Drop-17	25.6%	20.4%	73.7$_{-11.7}$	32.1$_{-28.2}$	62.3$_{-8.6}$	83.3$_{-8.2}$	79.5$_{-13.0}$	73.0$_{-14.0}$	71.8$_{-10.3}$	68.0$_{-13.4}$

J Full per-task results on RoboTwin 2.0

Table 17 provides the full per-task numerical results corresponding to Figure 4. We report success rates (%) for Easy (clean) and Hard (randomized) evaluation variants on each of the 7 RoboTwin 2.0 tasks under different drop levels.

Table 17: Full per-task results on RoboTwin 2.0 with $\pi_{0.5}$ (**`GateProbe`** block selection). Success rate (%) for Easy and Hard variants across drop levels. Avg is the mean of Easy and Hard.

	Baseline			Drop-9			Drop-12			Drop-16		
Task	Easy	Hard	Avg	Easy	Hard	Avg	Easy	Hard	Avg	Easy	Hard	Avg
Beat Block Hammer	32	16	24.0	37	0	18.5	30	0	15.0	25	3	14.0
Click Bell	68	50	59.0	54	47	50.5	21	28	24.5	33	48	40.5
Lift Pot	20	18	19.0	26	3	14.5	19	4	11.5	29	8	18.5
Move Playingcard	70	44	57.0	62	31	46.5	46	17	31.5	63	10	36.5
Open Microwave	82	60	71.0	90	55	72.5	65	15	40.0	85	8	46.5
Place Dual Shoes	2	0	1.0	2	0	1.0	3	0	1.5	4	0	2.0
Turn Switch	18	8	13.0	17	14	15.5	12	17	14.5	17	17	17.0
Average	41.7	28.0	34.9	41.1	21.4	31.3	28.0	11.6	19.8	36.6	13.4	25.0

K FULL COMPRESSION COMPARISON ON LIBERO-GOAL

Here we report the full 12-method comparison on OpenVLA-OFT / LIBERO-Goal (10 tasks $\times$ 5 trials = 50 episodes, max 300 steps, seed 7), including INT4 quantization and Wanda 2:4 pruning (Sun et al., 2023) that are omitted from the main text.

Setup. The Dense Baseline and DTR results are taken from the OpenVLA-OFT rows in Table 1. For CosSim-based zero-shot dropping, we profile the baseline on LIBERO 4-suite calibration data (64 batches $\times$ 8 = 512 samples) to compute Block Influence (BI) $= 1 - \cos(\mathbf{h}_{\text{in}}, \mathbf{h}_{\text{out}})$ at block, attention, and MLP granularity. Layers with lowest BI (closest to identity) are dropped first. Wanda 2:4 checkpoints are created from baseline activation norms. Latency and memory are measured on a single H200 GPU using the inference pipeline (VLA forward + action head prediction, batch size 1, 32 iterations, 5-iteration warmup, `torch.cuda.synchronize`).

Table 18: Full comparison: 12 compression methods on OpenVLA-OFT / LIBERO-Goal. "Step Ratio" compares total environment steps (including failed episodes at 300 max steps) against the baseline. Task Speedup = Act. Speedup / Step Ratio.

Method	SR (%) ↑	Act. Speedup ↑	Latency ↓	Memory ↓	Step Ratio ↓	Task Speedup ↑
Dense Baseline	98.0	1.00×	225.1	14.40	1.00×	1.00×
Trained block drop						
DTR-16	**100.0**	1.56×	144.4	8.36	**0.95**×	1.64×
DTR-30	90.0	**2.94**×	**76.7**	**3.06**	1.13×	**2.60**×
Traditional compression (applied to baseline, no retraining)						
INT4 Quantization	94.0	0.61×	369.5	4.71	1.10×	0.55×
Wanda 2:4 (LLM only)	42.0	0.99×	226.8	14.40	1.95×	0.51×
Wanda 2:4 (full)	28.0	1.03×	217.5	14.40	2.24×	0.46×
CosSim zero-shot drop (applied to baseline, no retraining)						
Block Drop 4	78.0	1.05×	214.4	12.89	1.46×	0.72×
Block Drop 8	18.0	1.31×	171.5	11.38	2.39×	0.55×
Attn Drop 4	**100.0**	1.09×	206.1	13.90	0.99×	1.10×
Attn Drop 8	98.0	1.18×	191.5	13.40	1.08×	1.09×
MLP Drop 4	0.0	1.15×	196.1	13.40	2.71×	0.42×
MLP Drop 8	0.0	1.19×	188.6	12.39	2.71×	0.44×

Key observations. *DTR is the only method that improves all three axes.* DTR-16 achieves 100% SR (+2.0 over baseline), 1.64× task speedup, and 42% memory savings. No other method improves even two axes simultaneously.

Per-action speedup does not imply end-to-end speedup. We report two speedup measures: *Act. Speedup*, the per-action inference speedup; and *Task Speedup* = Act. Speedup / Step Ratio, the end-to-end speedup for completing all evaluation episodes. Methods that degrade SR inflate total environment steps (failed episodes run to the 300-step horizon), so a faster per-action model can be *slower* overall. For example, Block Drop 4 achieves 1.05× action speedup but only 0.72× task speedup, and Wanda 2:4 (full) achieves 1.03× action speedup but only 0.46× task speedup. In contrast, DTR-16 combines 1.56× action speedup with a favorable step ratio (0.95×), yielding 1.64× task speedup. This shows that recovery training is not merely beneficial but necessary: without it, step overhead from degraded SR more than offsets per-action gains.

Traditional compression underperforms in this kernel-agnostic setting. All methods are benchmarked on a standard H200 environment *without* specialized quantization or sparsity kernels. Under this setting, Wanda 2:4 pruning drops to 42% (LLM-only) or 28% (full-model) SR with no speedup or memory benefit, and INT4 quantization preserves 94% SR and saves memory but is *slower* (0.61× action, 0.55× task) due to dequantization overhead. These results reflect the absence of optimized runtime support; with dedicated low-bit or structured-sparsity kernels, quantization and pruning methods can achieve substantially better latency.

Attention sublayers are highly redundant; MLP sublayers are not. Zero-shot CosSim-based attention dropping removes up to 8 attention sublayers (25% of all attention) with no SR loss (Attn Drop

8: 98%, task speedup 1.09×), confirming large attention redundancy. In stark contrast, removing even 4 MLP sublayers causes complete failure (0% SR, task speedup 0.42×). The MLP BI scores have 10× lower variance than attention BI scores, yet the cosine-similarity metric fundamentally underestimates MLP criticality. This asymmetry explains why zero-shot *block* dropping (which removes both attention and MLP) performs poorly (Block Drop 4: 78%, task speedup 0.72×), and why DTR's fine-tuning phase is essential to recover from MLP removal.

L EDGE ACCELERATION REQUIRES HARDWARE-KERNEL ALIGNMENT

A practical motivation for **DTR** is that reducing parameter count or theoretical FLOPs does not necessarily produce proportional wall-clock speedups on edge robotic platforms. In deployment, a compressed model must match both the hardware capabilities of the target device and the optimized kernels provided by the inference runtime. This is especially important for closed-loop robotic control, where latency affects control bandwidth and end-to-end task completion time.

Low-bit quantization. Low-bit quantization can reduce memory bandwidth and increase arithmetic throughput, but only when the target runtime supports the corresponding quantized operators. For example, Jetson Thor exposes Blackwell-class low-precision compute, including FP4 capability, while TensorRT supports quantized formats such as INT8, INT4, FP8, and FP4 (NVIDIA, 2026a;b). However, a GPTQ- or INT4-compressed model does not automatically become faster on an edge device: the relevant linear, attention, and normalization operators must be lowered into efficient low-bit kernels. Otherwise, dequantization overhead, unsupported operators, or fallback execution can reduce or eliminate the expected speedup.

Sparse pruning. Sparsity-based pruning has an analogous constraint. Hardware acceleration for sparsity typically assumes a specific structured pattern rather than arbitrary zeros. NVIDIA sparse Tensor Cores, for instance, target fine-grained 2:4 structured sparsity, where two values in each group of four are zero (Bai & Li, 2023). TensorRT also requires sparse weights to satisfy the supported structure and to be executed under compatible precision modes (NVIDIA, 2026c). As a result, generic unstructured pruning, or sparse weights that do not match the required pattern, may reduce model size without yielding reliable latency gains on Jetson-class devices.

Implication for DTR. DTR avoids this hardware-kernel alignment issue by physically removing transformer blocks and producing a smaller standard dense model. The resulting network uses ordinary dense operators with fewer layers, so its speedup does not rely on low-bit arithmetic, sparse Tensor Cores, or specialized sparse kernels. This is why we treat hardware friendliness separately from theoretical FLOPs reduction: quantization and sparsity can be effective when the deployment stack supports them, but DTR provides a simpler and more portable path to acceleration on edge robotic systems.