SPARSEVLA: TOWARDS SMALLER AND FASTER VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) models offer a promising approach to general-purpose robotic control, but their large parameter counts contribute to substantial storage requirements and inference latency, hindering efficient deployment. Weight pruning is a representative approach to addressing this problem. However, when applied to VLA models, existing weight pruning methods typically use information from each control step in isolation, overlooking the sequential nature of VLA control. In practice, two challenges arise: 1) Methods may reconstruct individual outputs well but fail to preserve how they change across steps; 2) Sparse execution can slow down inference when runtime overheads outweigh computational savings. To solve these challenges, we introduce **SparseVLA**, a training-free framework for hardware-efficient VLA weight pruning and deployment. We find that activation changes between steps reflect task progression, suggesting their potential as a complementary signal for weight pruning. Since preserving inter-step activation changes can favor different pruning patterns than reconstructing individual outputs, SparseVLA balances the two criteria when selecting and reconstructing weights. To turn this sparsity into practical inference gains, SparseVLA combines shape-aware sparse execution with operator fusion, reducing memory traffic and kernel launch overhead. Across LIBERO, SimplerEnv, and real-world manipulation tasks, SparseVLA maintains or improves aggregate task success relative to dense baselines on two VLA backbones, $\pi_{0.5}$ and GR00T, while reducing model footprint and inference latency.

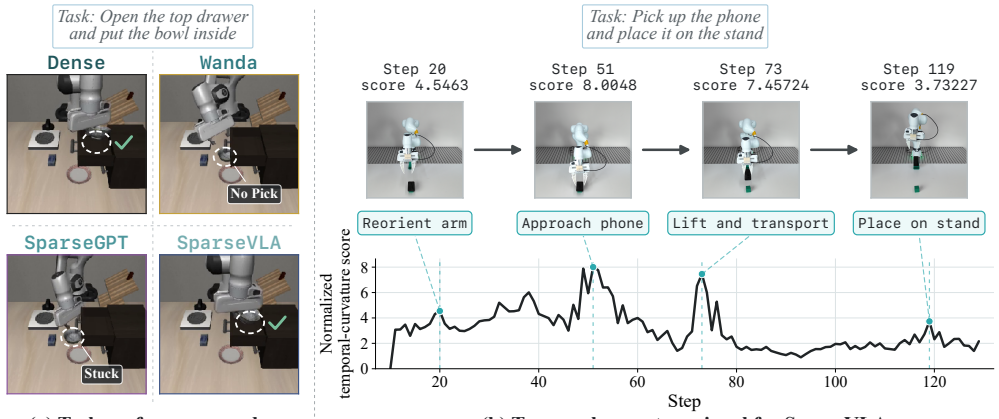
1 INTRODUCTION

Vision-language-action (VLA) models enable general-purpose robotic control (Black et al., 2025; NVIDIA et al., 2025), but their large parameter counts contribute to substantial storage requirements and inference latency, hindering efficient deployment. For example, $\pi_{0.5}$ contains approximately 3.62 billion parameters, requiring over 7 GB for BF16 weights. Recent methods reduce inference latency through token pruning, feature caching, model scheduling, or more efficient action generation (Jiang et al., 2025; Xu et al., 2025; Li et al., 2026; Pertsch et al., 2025; Ji et al., 2026a;c;b), but generally leave the weight storage requirements unchanged.

Weight pruning offers a complementary approach to model compression, with hardware-supported patterns such as 2:4 semi-structured sparsity offering the potential for GPU acceleration (Mishra et al., 2021; Lin et al., 2023). In VLA control, a policy acts step by step, with each action affecting the observation used at the next step. Consistent with this dependence, we observe that pruning-induced action errors can compound during task execution (Figure 2(a)). Many existing methods, however, prune weights by using information extracted from different steps independently, without explicitly considering the relationships between these steps (Frantar & Alistarh, 2022; 2023; Kuznedelev et al., 2023). Moreover, at the same sparsity level, two pruned models can have similar output errors at individual steps but differ in how well they preserve changes between steps. This motivates an additional pruning criterion: *preserving changes across control steps*, which can be used to guide weight selection and help to limit error accumulation during task execution.

To incorporate this criterion, we draw on the temporal structure already present in calibration trajectories. In Figure 1(b), activation changes align with key stages of task execution and trace how the model's responses evolve as the task unfolds. We seek to preserve these changes in responses

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.



(a) Task performance under different pruning methods

(b) Temporal-curvature signal for SparseVLA coincides with salient rollout moments

Figure 1: **Closed-loop outcomes and temporal-curvature responses aligned with task progress.** (a) The four policies start from the same initial state in the illustrated LIBERO task. Dense and SparseVLA complete the task, whereas Wanda fails to pick up the bowl and SparseGPT stalls during execution. (b) SparseVLA's normalized temporal-curvature score summarizes stabilized inter-step activation changes through a parameter-normalized weighted sum across target projections in all layers of $\pi_{0.5}$, with local peaks corresponding to key stages of task execution.

between consecutive steps. However, matching these differences alone allows a constant offset from the dense outputs. We therefore retain the output reconstruction constraint. Together, these criteria guide pruning to preserve both layer outputs and their evolution along calibration trajectories.

Building on these insights, we introduce **SparseVLA**, a training-free framework for hardware-efficient VLA weight pruning and deployment. SparseVLA innovatively evaluates each of the six valid 2:4 patterns as a whole, explicitly accounting for interactions between the weights removed together. It first retains candidates within a local output-reconstruction budget, then scores the remaining patterns by their estimated error in reconstructing both layer outputs and inter-step changes. With the mask fixed, SparseVLA compensates for pruning error by blending two weight reconstructions, one matching individual layer outputs and the other also matching their changes across control steps.

To translate this 2:4 sparsity into storage and latency savings, we also optimize how the pruned weights are stored and executed. SparseVLA introduces shape-aware sparse execution, applying packed 2:4 kernels only where they provide a net benefit and retaining efficient dense execution elsewhere. It also fuses compatible projections with their activation–product operations. This reduces intermediate memory traffic while preserving the selected masks.

We evaluate SparseVLA on $\pi_{0.5}$ and GR00T (Black et al., 2025; NVIDIA et al., 2025) across simulation and real-robot settings. SparseVLA achieves 97.25% versus Dense's 96.15% on 40 LIBERO tasks and 65.77% versus 63.15% across Google Robot and WidowX. On four manipulation tasks using a single Franka Research 3 (FR3) arm, the deployed sparse policies match Dense at $59/80$ successes. On $\pi_{0.5}$, exact 2:4 pruning removes 1.147 billion weights, saving 21.9% of checkpoint size and 21.1% of peak CUDA allocation. Our sparse deployment achieves a $1.215\times$ speedup for $\pi_{0.5}$ within the same optimized backend, reducing policy latency by 17.7%. Combining backend optimization with sparse execution yields final policy-inference latencies of only 32.85 ms for $\pi_{0.5}$, 25.04 ms for GR00T on Google Robot, and 24.33 ms for GR00T on WidowX.

Our contributions are summarized as follows: **(1)** We show that pruning based only on output reconstruction can produce action errors that accumulate across control steps. Including activation changes across steps can also alter which weights are retained, with the rest of the pruning procedure unchanged. **(2)** We propose SparseVLA, a training-free method that evaluates complete 2:4 patterns to account for interactions between weights removed together. It uses activation changes across steps to select patterns within a local output-reconstruction budget and adjust the retained weights under

the selected mask. **(3)** We develop shape-aware sparse execution with mask-preserving operator fusion to translate hardware-compatible sparsity into memory reductions and practical latency.

2 Related Work

2.1 Efficient Vision-Language-Action Models

RoboMamba and TinyVLA adopt state-space and compact multimodal backbones, respectively (Liu et al., 2024; Wen et al., 2025), while FAST uses frequency-space action tokenization (Pertsch et al., 2025). Reusing visual representations or intermediate denoiser features (Xu et al., 2025; Ji et al., 2026a), pruning visual tokens (Pei et al., 2026), and scheduling models (Li et al., 2026) reduce redundant computation. EfficientVLA combines language-layer pruning, visual-token selection, and diffusion-head caching (Yang et al., 2025).

2.2 Weight Pruning Methods for Large Models

Weight-pruning methods differ in weight selection and error compensation. Optimal Brain Surgeon uses inverse-Hessian compensation, and SparseGPT extends second-order reconstruction to one-shot LLM pruning (Hassibi & Stork, 1992; Frantar & Alistarh, 2023), whereas Wanda and D^{2}Prune use weight–activation saliency and joint perturbation modeling, respectively (Sun et al., 2024; Xiong et al., 2026). MaskLLM learns $N{:}M$ masks (Fang et al., 2024), while EfficientLLaVA combines sparsity allocation with OBS-based pruning for vision-language models (Liang et al., 2025). VLA pruning must preserve closed-loop task performance. GLUESTICK applies training-free low-rank correction after semi-structured pruning (Jabbour et al., 2025). EcoVLA combines environment-adaptive channel pruning with interleaved inference orchestration (Huang et al., 2026), while VLM-to-VLA parameter divergence guides module-aware pruning (Zhang et al., 2026).

2.3 Sparse Inference and Runtime Optimization

Weight sparsity requires hardware and runtime support to reduce latency. Sparse Tensor Cores support 2:4 sparsity, while nmSPARSE enables broader $N{:}M$ execution (Mishra et al., 2021; Lin et al., 2023). CUTLASS and cuSPARSELt provide structured-sparse matrix multiplication primitives (NVIDIA Corporation, 2024; n.d.). RealtimeVLA reduces inference overhead through execution simplification and operator fusion (Ma et al., 2025).

3 SparseVLA

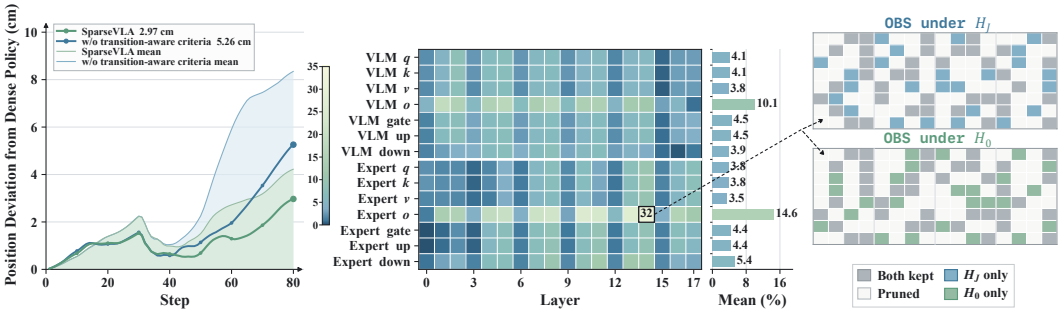
3.1 Why Calibration-Set Output Fidelity Alone Can Be Insufficient

Conventional reconstruction-based pruning matches layer outputs on recorded observations. For layer ℓ, let W^ℓ denote the dense weights, $\widetilde{W}^\ell_M$ the pruned weights under mask M, and $X^\ell_{e,t}$ the input activations at recorded control step t of rollout e. With $\mathcal{C}^\ell$ indexing valid calibration steps, the output-reconstruction objective is

$$\mathcal{L}^\ell_{\text{out}}(M) = \sum_{(e,t)\in\mathcal{C}^\ell} a_{e,t} \left\| \left(W^\ell - \widetilde{W}^\ell_M\right) X^\ell_{e,t} \right\|_F^2, \tag{1}$$

where $a_{e,t} \geq 0$ weights calibration samples. For a fixed collection of weighted activations, this objective is unchanged by their temporal ordering. It measures the responses at individual observations without explicitly using the transitions between them.

Finding 1: Output reconstruction can leave growing closed-loop deviations. Figure 2(a) illustrates this gap under output-fidelity-only pruning. Starting from the same initial state, the pruned rollout departs from the dense-policy trajectory, reaching an end-effector position deviation of 5.26 cm in the illustrated example. The per-step mean over three paired rollouts of the same task shows a similar overall growth in deviation.



(a) Deviation from the dense-policy rollout (b) OBS mask disagreement (%) (c) 2:4 masks at Expert o, layer 14

Figure 2: **Closed-loop deviations and the effect of temporal information on pruning preferences. (a)** End-effector position deviations from the dense policy for the H_0-only baseline and full SparseVLA. Thick curves show an example rollout; thin curves show per-step means over three paired rollouts of the same task, with the area below each mean shaded. **(b)** Mask disagreement between OBS under H_0 and OBS under H_J, using the same 2:4 pattern-selection procedure. **(c)** The corresponding retained-weight patterns in one projection.

A local analysis explains how such deviations can develop. Let $z_{t+1} = F(z_t, \pi(z_t))$ denote the dense closed loop, and let $\widetilde{\pi}$ denote the pruned policy. Define the state deviation $\delta z_t = \widetilde{z}_t - z_t$ and the local policy error $\varepsilon_t = \widetilde{\pi}(z_t) - \pi(z_t)$. A first-order expansion around the dense trajectory, neglecting pruning-induced Jacobian changes and higher-order terms, gives

$$\delta z_{t+1} \approx A_t \delta z_t + B_t \varepsilon_t, \tag{2}$$

where $A_t = \partial_z F(z, \pi(z))|_{z_t}$, $B_t = \partial_a F(z_t, a)|_{a=\pi(z_t)}$. Local action errors can be amplified by subsequent dynamics, while existing state deviations change the inputs encountered by the policy. Reconstruction on recorded observations does not explicitly account for this feedback. This makes the policy's response to evolving observations relevant in addition to its fit at recorded states.

Finding 2: Temporal information changes pruning preferences. Calibration trajectories contain both layer responses and the changes linking successive responses. To characterize the latter, consider valid within-trajectory transitions $(e, t) \in \mathcal{T}^\ell$, with simulator-step gap $d_{e,t}$, and let $\Delta X^\ell_{e,t} = (X^\ell_{e,t} - X^\ell_{e,t-1})/d_{e,t}$. For a linear layer, the discrepancy between dense and pruned inter-step output changes is $(W^\ell - \widetilde{W}^\ell_M)\Delta X^\ell_{e,t}$. The corresponding fidelity measure is

$$\mathcal{L}^\ell_{\text{step}}(M) = \sum_{(e,t)\in\mathcal{T}^\ell} b_{e,t} \left\| \left(W^\ell - \widetilde{W}^\ell_M\right) \Delta X^\ell_{e,t} \right\|_F^2, \tag{3}$$

where $b_{e,t} \geq 0$ weights transitions. Writing the output residual as $E^\ell_{e,t} = (W^\ell - \widetilde{W}^\ell_M)X^\ell_{e,t}$, the residual in Equation 3 is $(E^\ell_{e,t} - E^\ell_{e,t-1})/d_{e,t}$. Thus, output fidelity measures residual magnitudes, whereas inter-step fidelity measures their variation along recorded transitions. Residual sequences with the same pointwise magnitudes can have different inter-step variations, so similar output-reconstruction errors need not imply similar preservation of response changes.

This distinction also appears in actual pruning decisions. We compare correlated OBS under output-fidelity curvature H_0 and curvature H_J, which additionally incorporates transition statistics. Both runs start from the same dense checkpoint and calibration observations and follow the same sequential OBS procedure over the six legal 2:4 removal patterns. STEP is disabled in both, and the output-fidelity trust budget is fixed. Figure 2(b) shows that mask disagreement averaged over 18 layers ranges from 3.5–14.6% across subsystem–projection categories; the layer-14 action-expert o-projection reaches 31.6%. Panel (c) visualizes the corresponding retained-weight patterns in one projection (protocol in Appendix B). The comparison shows that incorporating temporal statistics changes which weights are favored under the same sparsity constraint.

Inter-step fidelity remains complementary to output fidelity: a constant nonzero residual along a trajectory cancels under differencing, despite leaving the layer outputs biased. The two criteria

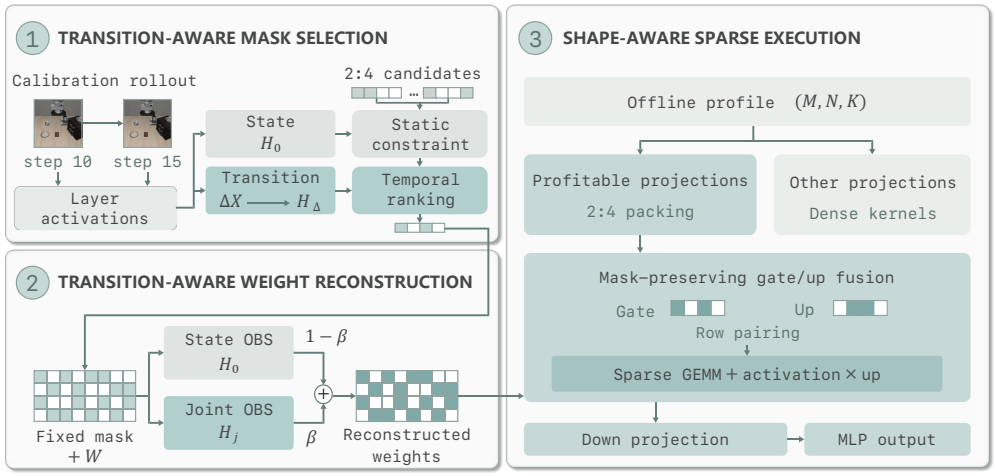


Figure 3: **Overview of SparseVLA. Top left:** Transition-aware 2:4 mask selection with a static-cost constraint and temporal ranking. **Bottom left:** State and joint-curvature OBS reconstructions are blended under a fixed mask. **Right:** Offline shape profiling routes projections to sparse or dense execution. Pairing gate and up rows enables fused activation–product before the down projection, preserving the selected 2:4 masks.

therefore address distinct aspects of response preservation. Temporal information can distinguish how sparse approximations preserve response changes, while output fidelity constrains the response values themselves.

3.2 Transition-Aware Mask Selection

Figure 3 summarizes the SparseVLA pipeline, combining transition-aware mask selection, fixed-mask weight reconstruction, and hardware-efficient sparse execution.

SparseVLA applies 2:4 sparsity along each linear layer's input dimension. It retains rollout order and computes differences only within a trajectory. We accumulate the output-fidelity and transition curvatures

$$H_0^\ell = \sum_{(e,t)\in\mathcal{C}^\ell} a_{e,t} X_{e,t}^\ell (X_{e,t}^\ell)^\top, \qquad H_\Delta^\ell = \sum_{(e,t)\in\mathcal{T}^\ell} b_{e,t} \Delta X_{e,t}^\ell (\Delta X_{e,t}^\ell)^\top. \tag{4}$$

Up to normalization, these are the curvatures of Equations 1 and 3. SparseVLA uses H_0^ℓ to identify reliable patterns and transition statistics to choose among them.

Correlated Pattern Scoring. For one four-weight group in row i, let $\mathcal{P} = \{S \subset \{1,2,3,4\} : |S| = 2\}$ be its six legal removal patterns. Element-wise ranking misses interactions between the two removals. For curvature H and the current OBS-compensated group w_H, we score the complete pattern as

$$C_H(S) = (w_H)_S^\top \left((H^{-1})_{SS}\right)^{-1} (w_H)_S, \qquad S \in \mathcal{P}. \tag{5}$$

Up to a constant, this is the minimum quadratic reconstruction loss subject to zeroing S, including the off-diagonal correlation discarded by summed element-wise OBS scores (Hassibi & Stork, 1992; Frantar & Alistarh, 2023). The curvature H specifies the statistics used by this generic pattern cost: C_{H_0} defines the output-fidelity trust set, while C_{H_J} scores patterns for final selection under the joint curvature (Equation 7). Thus, the final OBS score incorporates inter-step information; STEP adds a first-order temporal signal. After each group, we propagate its correction to later columns separately under the two curvatures.

Output-Fidelity Trust Set. Finite-difference statistics can be noisy, so transition fidelity should not incur an uncontrolled output error. We evaluate all six patterns under H_0^ℓ and admit only

$$\mathcal{A}_\tau = \left\{ S \in \mathcal{P} :\ C_{H_0}(S) \le C_{H_0}^{\min} + \tau\left(C_{H_0}^{\max} - C_{H_0}^{\min}\right) \right\}, \tag{6}$$

where $\tau \in [0, 1]$. The normalized budget is group-adaptive: $\tau = 0$ admits only output-optimal patterns, while $\tau = 1$ admits all six.

Transition-Aware Selection. We form a joint curvature

$$H_J^\ell = H_0^\ell + \lambda H_\Delta^{\ell,\mathrm{stab}}. \tag{7}$$

Because H_Δ^ℓ can differ greatly in scale from H_0^ℓ, we first scale its off-diagonal component by $\kappa \in [0, 1]$, then trace-match it under a scale cap and bound it coordinate-wise and spectrally. Specifically, with $D_\Delta^\ell = \mathrm{Diag}(\mathrm{diag}(H_\Delta^\ell))$, we replace H_Δ^ℓ by $D_\Delta^\ell + \kappa(H_\Delta^\ell - D_\Delta^\ell)$. With $D_0^\ell = \mathrm{Diag}(\mathrm{diag}(H_0^\ell))$, we then cap $\lambda\,\mathrm{diag}(H_\Delta^{\ell,\mathrm{stab}})$ relative to $\mathrm{diag}(H_0^\ell)$ and cap the largest eigenvalue of $\lambda(D_0^\ell)^{-1/2} H_\Delta^{\ell,\mathrm{stab}} (D_0^\ell)^{-1/2}$. This preserves positive semidefiniteness while preventing transition curvature from dominating the static scale.

Transition curvature captures correlations between changing input directions. STEP saliency adds a first-order signal for weights connected to rapidly changing channels:

$$G_{ij}^\ell = \left|W_{ij}^\ell\right| \left(s_i^{\mathrm{out}} + s_j^{\mathrm{in}}\right), \tag{8}$$

where s_i^{out} and s_j^{in} are transition-weighted mean absolute changes in channelwise activation norms. After matching its within-group scale to the OBS cost, yielding $\widehat{G}_{ij}^\ell$, we select

$$S^* = \underset{S \in \mathcal{A}_\tau}{\arg\min}\, C_{H_J}(S) + \gamma \sum_{j \in S} \widehat{G}_{ij}^\ell. \tag{9}$$

The trust set bounds the output-fidelity cost of both transition signals.

3.3 Transition-Aware Weight Reconstruction

Mask selection fixes the zeros, but direct removal leaves avoidable error. Let Ω collect a layer's zero positions. We keep Ω fixed and perform blockwise OBS reconstruction under both curvatures:

$$W_q^* = \underset{\widehat{W}:\, \widehat{W}_\Omega = 0}{\arg\min}\ \mathrm{Tr}\Big[(\widehat{W} - W) H_q (\widehat{W} - W)^\top\Big], \qquad q \in \{0, J\}. \tag{10}$$

The transition-aware solution W_J^* uses the same H_J as final pattern selection, so inter-step statistics guide both which weights are retained and how they compensate for pruning error. SparseVLA blends it with the output-fidelity solution W_0^*:

$$\widetilde{W} = (1 - \beta) W_0^* + \beta W_J^*, \qquad 0 \le \beta \le 1. \tag{11}$$

Both endpoints share the fixed-mask subspace, so interpolation preserves the mask; we reapply it to guarantee exact 2:4 sparsity. The procedure is training-free; trajectory statistics are used only offline and discarded after pruning.

3.4 Hardware-Efficient SparseVLA Inference

Exact 2:4 zeros alone save neither storage nor latency when checkpoints and GEMMs remain dense. We translate SparseVLA's pruning into deployment gains through weight packing, kernel selection and sparse operator fusion.

Why Sparse Execution Can Be Slower. 2:4 sparse GEMMs reduce arithmetic but require sparsity metadata and may incur activation padding and output-layout copies. Padding also adds computation on unused entries. For short sequences or small projections, these costs can outweigh the saved computation, making sparse execution slower than dense execution. Whether sparsity reduces latency therefore depends on the actual matrix shape.

Shape-Aware Sparse Execution. We therefore compare dense and sparse execution at each projection's actual (M, N, K) shape and pack its retained values and 2:4 metadata only when the sparse path is faster. Separate gate and up kernels also write intermediate outputs that the activation–product operation must read back. To remove this traffic, we pair gate and up rows while preserving their independent 2:4 masks. One sparse GEMM computes both projections and fuses the

activation–product into its epilogue, writing directly in the layout needed by the down projection. This extends the execution plan without changing the pruning decisions. For $\pi_{0.5}$, we build on RealtimeVLA (Ma et al., 2025) and extend its backend with shape-aware 2:4 execution and mask-preserving sparse operator fusion. Selected VLM gate, up, and down projections run sparsely; the short action expert remains dense. GR00T uses sparse gate and up projections with SiLU fusion and dense attention and down projections.

4 Experiments

We evaluate SparseVLA along three dimensions: closed-loop manipulation success, model footprint, and deployed inference efficiency. Experiments cover two VLA families in simulation, physical robot rollouts, and ablations of the pruning and execution components.

4.1 Experimental Setup

Models and benchmarks. We test $\pi_{0.5}$ (Black et al., 2025) on 40 LIBERO tasks (Liu et al., 2023) (50 rollouts per task) and GR00T N1.7 (NVIDIA et al., 2025) on six Google Robot and seven WidowX SimplerEnv tasks (Li et al., 2025) (100 rollouts per task). Task-weighted GR00T averages combine Google and WidowX in a 6:7 ratio; each embodiment shares one checkpoint. We also test four task-specific GR00T N1.6 policies on one Franka Research 3 (FR3) arm (Section 4.3).

Pruning and baselines. Baselines are magnitude pruning, Wanda (Sun et al., 2024), SparseGPT (Frantar & Alistarh, 2023), and GLUESTICK (Jabbour et al., 2025), using SparseGPT-pruned checkpoints with rank-200 low-rank correction for the latter. All training-free methods prune the same matrices to exact 2:4. SparseVLA retains rollout order, Appendix C defines the pruning hyperparameters and lists their default values.

Efficiency measurement. Measurements use RTX 4090 GPUs, batch size one and BF16, excluding initialization, compilation and warmup. Success rates and memory savings in Table 1 are measured in eager mode. Additional closed-loop checks with the same pruned weights show negligible differences between eager execution and the deployed runtime. The selective columns in Table 2(a) and panel (c) report speedups over unpruned Dense eager: CUDA-event medians in (a) and policy wall-time medians in (c). The full-sparse columns in (a) report complete-policy CUDA-event median speedups over unpruned Dense with the same official compilation settings. Panel (b) reports synchronized policy wall-time means and cross-run ratios. Appendix A gives timing boundaries, matched controls and operator ablations. Results in Tables 1–4 are rounded to the displayed precision; percentage-point gains are computed before rounding, and success counts are reported exactly.

4.2 Performance and Efficiency across VLA Models

Preserving manipulation performance. SparseVLA achieves the highest average success among the evaluated sparse methods on both model families (Table 1). On LIBERO, it reaches 97.25%, exceeding Dense by 1.10 percentage points and matching GLUESTICK without its 331.8M recovery coefficients. On GR00T, it reaches 65.77%, exceeding Dense, SparseGPT, and GLUESTICK by 2.62, 12.85, and 9.38 points, respectively. The gains over Dense hold for both Google Robot and WidowX. These results demonstrate that exact 2:4 pruning can retain strong manipulation performance across the two VLA families.

Model footprint. SparseVLA reduces stable eager CUDA allocation by 20.95% for $\pi_{0.5}$ and 5.23% for GR00T. For $\pi_{0.5}$, peak allocation, checkpoint size, and resident weights decrease by 21.1%, 21.92%, and 21.22%, respectively. At the same 97.25% LIBERO success rate, SparseVLA saves 20.95% of stable CUDA allocation compared with GLUESTICK's 12.16%. These whole-policy savings reflect the packed projections alongside the remaining dense weights, metadata, and runtime allocations.

Deployment efficiency. Within the same optimized backend, SparseVLA reduces $\pi_{0.5}$ policy latency by 17.7%, from 39.91 to 32.85 ms (1.215× speedup). GR00T achieves 1.026–1.029× policy

Table 1: **Success and memory savings (%).** All results are measured in eager mode. Bold: best sparse success. Memory: stable CUDA allocation saved vs. Dense. G/W: Google/WidowX.

	$\pi_{0.5}$ / LIBERO						GR00T N1.7 / SimplerEnv			
Method	Spat.	Obj.	Goal	Long	Avg.	Mem.	Google	WidowX	Avg.	Mem. (G/W)
Dense	98.0	98.4	97.2	91.0	96.15	–	65.17	61.43	63.15	– / –
Magnitude	81.4	89.6	41.8	19.8	58.15	20.95	14.50	3.29	8.46	5.23 / 5.23
Wanda	95.0	98.4	89.6	77.8	90.20	20.95	41.00	38.71	39.77	5.23 / 5.23
SparseGPT	97.0	**99.8**	97.4	92.2	96.60	20.95	57.67	48.86	52.92	5.23 / 5.23
GLUESTICK	98.0	**99.8**	**98.0**	**93.2**	**97.25**	12.16	57.83	55.14	56.38	1.50 / 1.27
SparseVLA	**98.2**	**99.8**	97.8	**93.2**	**97.25**	20.95	**67.50**	**64.29**	**65.77**	5.23 / 5.23

Table 2: **Inference efficiency on RTX 4090.** (a) Speedups over matching Dense: selective sparse execution in eager mode (E) and full-sparse execution with official compilation (C). (b) Historical compiled reference vs. final deployment (mean ms). (c) Robot speedups. G/W: Google/WidowX.

(a) Inference speedup (↑)

	Selective (E)			Full-sparse (C)		
Method	$\pi_{0.5}$	G	W	$\pi_{0.5}$	G	W
Magnitude	1.021	1.008	0.999	0.684	0.876	0.874
Wanda	1.024	1.000	1.002	0.666	0.886	0.876
SparseGPT	1.019	0.998	1.000	0.666	0.877	0.876
GLUESTICK	0.779	0.940	0.940	0.552	0.785	0.783
SparseVLA	1.018	1.009	1.004	0.664	0.879	0.882

(b) Historical / final

Model	Ref.	Ours	Ratio (↑)
$\pi_{0.5}$	69.72	32.85	2.12
GR–G	52.20	25.04	2.08
GR–W	57.25	24.33	2.35

(c) Robot speedup (↑)

Phone	Pen	Bowls	Bun
1.92	2.01	2.07	1.91

speedups over dense execution of the same pruned weights (Appendix A). Combining backend optimization with sparse execution yields final policy-inference latencies of 32.85 ms for $\pi_{0.5}$, 25.04 ms for GR00T on Google Robot, and 24.33 ms for GR00T on WidowX (Table 2(b)).

Full-sparse execution can slow inference. Executing every pruned projection with standard 2:4 kernels slows all five methods relative to unpruned Dense under matched official compilation (Table 2(a), full-sparse columns). Speedups are 0.552–$0.684\times$ for $\pi_{0.5}$ and 0.783–$0.886\times$ for GR00T. Without our execution scheme, SparseVLA yields only $0.664\times$ on $\pi_{0.5}$, $0.879\times$ on Google Robot, and $0.882\times$ on WidowX. Reducing arithmetic alone therefore does not ensure lower policy latency, motivating shape-aware selection and fusion. Appendix A.1 gives the experimental protocol.

Language-backbone redundancy and deployment value. Preserving aggregate task success after 2:4 pruning reveals substantial compressible capacity in the evaluated VLAs. In $\pi_{0.5}$, the language backbone accounts for 86.4% of the removed weights, making it the main source of parameter reduction. One interpretation is that these manipulation workloads draw on a subset of the backbone's broadly pretrained capabilities, consistent with reported language-module redundancy (Yang et al., 2025). This motivates our focus on compressing language weights through trajectory-aware selection and reconstruction. The eager measurements in Table 2(a) also motivate pairing this compression with shape-aware dispatch and fusion to reduce launch and intermediate-memory overheads; realized policy speedups depend on the execution coverage (Appendix A). As language backbones grow, comparable prunable fractions would yield larger absolute weight savings, increasing the potential value of training-free compression for deploying larger VLAs.

4.3 Real-Robot Performance and Deployment

Robot platform and protocol. Dense and SparseVLA share one FR3 arm and control stack, with 20 trials per task and method (Figure 4). We time the final post-conversion SparseVLA policies against unpruned Dense eager using matched inputs (Table 2(c)). Appendix D provides task descriptions and additional toast and chocolate transfer sequences.

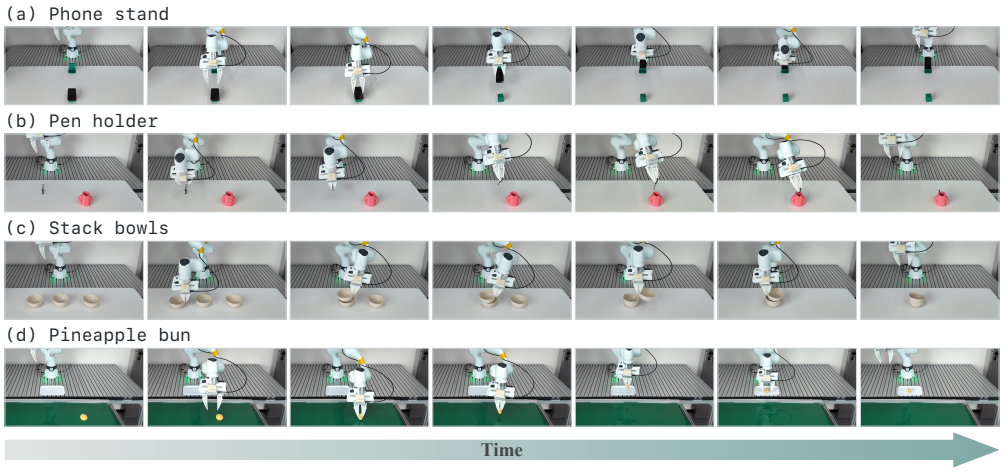


Figure 4: **Real-robot task sequences.** Seven frames per task, ordered left to right. Task order matches Table 3. Bowl stacking includes both transfers. Frames are not temporally aligned across tasks.

Table 3: **FR3 results.** Dense/SparseVLA (D/S): successes out of 20; peak memory saved vs. Dense eager.

Task	Success D / S	Mem. saved (%)
Phone stand	13 / 14	4.81
Pen holder	15 / 17	4.78
Stack bowls	12 / 10	4.78
Pineapple bun	19 / 18	4.77

Table 4: **Ablation success (%).**

Variant	$\pi_{0.5}$	GR00T
(a) Transition awareness		
Static ($\lambda = \gamma = 0$)	96.25	54.85
+ Mask	96.35	63.00
+ Reconstruction	97.25	65.77
(b) Pattern scoring		
Diagonal	96.75	59.62
Correlated	97.25	65.77

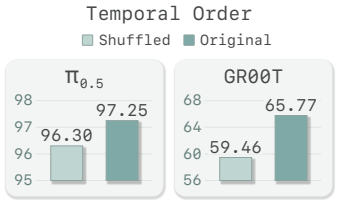


Figure 5: Success (%) with original and within-trajectory shuffled calibration order.

Success and deployment efficiency. The deployed SparseVLA policies match Dense at 59/80 successes (73.75%), while delivering 1.91–2.07× speedups across all four tasks and reducing peak CUDA memory by 4.77–4.81% (Tables 3 and 2(c)). Task-level success counts differ by one or two trials in either direction: phone placement and pen insertion improve, while bowl stacking and bun transfer decrease. The aggregate result demonstrates retained physical manipulation performance with a smaller and faster deployment.

4.4 Ablation Studies

With correlated scoring fixed, transition-aware mask selection raises success by 0.10 percentage points on $\pi_{0.5}$ and 8.15 on GR00T. Reconstruction adds another 0.90 and 2.77 points, respectively (Table 4(a)). In panel (b), both variants use transition-aware selection and reconstruction with fixed hyperparameters. Correlated scores removal pairs jointly, whereas Diagonal sums per-weight OBS costs; Correlated gains 0.50 points on $\pi_{0.5}$ and 6.15 on GR00T. Shuffling observations within calibration trajectories reduces success by 0.95 percentage points on $\pi_{0.5}$ and 6.31 on GR00T (Figure 5). We recompute transition curvature and STEP statistics after shuffling, keeping all other settings fixed. This drop supports the value of temporal ordering beyond individual observations.

5 Conclusion

We presented SparseVLA, a training-free framework that combines trajectory-aware 2:4 pruning with hardware-efficient execution. Across simulation and real-robot experiments, SparseVLA matches or exceeds dense policies in aggregate success while reducing GPU memory usage and inference latency. Matched backend comparisons confirm additional speedups from sparse execution. For larger language backbones, comparable prunable fractions would translate into greater absolute storage savings, suggesting a promising role for training-free compression in deploying larger VLAs on robots with limited resources.

AI USE STATEMENT

We used generative AI tools to assist with manuscript writing and editing, including improvements to wording, grammar, clarity, and organization, as well as LaTeX formatting. The authors reviewed and revised the AI-assisted content to ensure that it accurately reflects the methods and results. The authors take full responsibility for the accuracy and integrity of the final manuscript.

REFERENCES

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

Gongfan Fang, Hongxu Yin, Saurav Muralidharan, Greg Heinrich, Jeff Pool, Jan Kautz, Pavlo Molchanov, and Xinchao Wang. MaskLLM: Learnable semi-structured sparsity for large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 7736–7758. Curran Associates, Inc., 2024. doi: 10.52202/079017-0248. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/0e9a05f5ce62284c91e4a33498899124-Paper-Conference.pdf.

Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 4475–4488. Curran Associates, Inc., 2022. doi: 10.52202/068431-0323. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/1caf09c9f4e6b0150b06a07e77f2710c-Paper-Conference.pdf.

Elias Frantar and Dan Alistarh. SparseGPT: Massive language models can be accurately pruned in one-shot. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10323–10337. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/frantar23a.html.

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. Hanson, J. Cowan, and C. Giles (eds.), *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992. URL https://proceedings.neurips.cc/paper_files/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf.

Yuting Huang, Leilei Ding, Zhipeng Tang, Zenghuan Zhu, Jiajun Deng, Xinrui Lin, Shuo Liu, Haojie Ren, Jianmin Ji, and Yanyong Zhang. Environment-aware adaptive pruning with interleaved inference orchestration for vision-language-action models, 2026. URL https://arxiv.org/abs/2602.00780.

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, 2025. URL https://arxiv.org/abs/2510.08464.

Kangye Ji, Yuan Meng, Hanyun Cui, Ye Li, Jianbo Zhou, Shengjia Hua, Lei Chen, and Zhi Wang. Block-wise adaptive caching for accelerating diffusion policy. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 22805–22858,

2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/2712b17bb58ea5b2b65c45857b024744-Paper-Conference.pdf.

Kangye Ji, Yuan Meng, Jianbo Zhou, Ye Li, Chen Tang, and Zhi Wang. Test-time sparsity for extreme fast action diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9414–9423, June 2026b.

Kangye Ji, Jianbo Zhou, Yuan Meng, Ye Li, Hanyun Cui, and Zhi Wang. Sparse actiongen: Accelerating diffusion policy with real-time pruning, 2026c. URL https://arxiv.org/abs/2601.12894.

Titong Jiang, Xuefeng Jiang, Yuan Ma, Xin Wen, Bailin Li, Kun Zhan, Peng Jia, Yahui Liu, Sheng Sun, and Xianpeng Lang. The better you learn, the smarter you prune: Towards efficient vision-language-action models via differentiable token pruning, 2025. URL https://arxiv.org/abs/2509.12594.

Denis Kuznedelev, Eldar Kurtić, Elias Frantar, and Dan Alistarh. CAP: Correlation-aware pruning for highly-accurate sparse vision models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 28805–28831. Curran Associates, Inc., 2023. doi: 10.52202/075280-1254. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/5bd9fbb3a5a985f80c16ddd0ec1dfc43-Paper-Conference.pdf.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Oier Mees, Karl Pertsch, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating real-world robot manipulation policies in simulation. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3705–3728. PMLR, 06–09 Nov 2025. URL https://proceedings.mlr.press/v270/li25c.html.

Ye Li, Yuan Meng, Zewen Sun, Kangye Ji, Chen Tang, Jiajun Fan, Xinzhu Ma, Shu-Tao Xia, Zhi Wang, and Wenwu Zhu. SP-VLA: A joint model scheduling and token pruning approach for VLA model acceleration. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 38155–38184, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/4072543747a14bbed76284cf2c04b9e9-Paper-Conference.pdf.

Yinan Liang, Ziwei Wang, Xiuwei Xu, Jie Zhou, and Jiwen Lu. EfficientLLaVA: Generalizable auto-pruning for large vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9445–9454, June 2025.

Bin Lin, Ningxin Zheng, Lei Wang, Shijie Cao, Lingxiao Ma, Quanlu Zhang, Yi Zhu, Ting Cao, Jilong Xue, Yuqing Yang, and Fan Yang. Efficient GPU kernels for N:M-sparse weights in deep learning. In D. Song, M. Carbin, and T. Chen (eds.), *Proceedings of Machine Learning and Systems*, volume 5, pp. 513–525. Curran, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/file/a10deb4d5227a8ea307ea8ff3cb712f4-Paper-mlsys2023.pdf.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791. Curran Associates, Inc., 2023. doi: 10.52202/075280-1939. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/8c3c666820ea055a77726d66fc7d447f-Paper-Datasets_and_Benchmarks.pdf.

Jiaming Liu, Mengzhen Liu, Zhenyu Wang, Pengju An, Xiaoqi Li, Kaichen Zhou, Senqiao Yang, Renrui Zhang, Yandong Guo, and Shanghang Zhang. RoboMamba: Efficient vision-language-action model for robotic reasoning and manipulation. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 40085–40110. Curran Associates, Inc., 2024. doi: 10.52202/

079017-1266. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/46a126492ea6fb87410e55a58df2e189-Paper-Conference.pdf.

Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running VLAs at real-time speed, 2025. URL https://arxiv.org/abs/2510.26742.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks, 2021. URL https://arxiv.org/abs/2104.08378.

NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots, 2025. URL https://arxiv.org/abs/2503.14734.

NVIDIA Corporation. CUTLASS: CUDA templates for linear algebra subroutines. Software, version 3.5.1, 2024. URL https://github.com/NVIDIA/cutlass/tree/v3.5.1. Accessed: 2026-08-25.

NVIDIA Corporation. cuSPARSELt: A high-performance CUDA library for sparse matrix–matrix multiplication. Software documentation, n.d. URL https://docs.nvidia.com/cuda/cusparselt/. Accessed: 2026-08-25.

Xiaohuan Pei, Yuxing Chen, Siyu Xu, Yunke Wang, Yuheng Shi, and Chang Xu. Action-aware dynamic pruning for efficient vision-language-action manipulation. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 10832–10851, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/1223ca2b14513106ad55db6a537337f9-Paper-Conference.pdf.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. FAST: Efficient Action Tokenization for Vision-Language-Action Models. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025. doi: 10.15607/RSS.2025.XXI.012.

Mingjie Sun, Zhuang Liu, Anna Bair, and Zico Kolter. A simple and effective pruning approach for large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 4942–4964, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/14c856c7a41297804de4c4890e846b25-Paper-Conference.pdf.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, Yaxin Peng, Feifei Feng, and Jian Tang. TinyVLA: Toward fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 10(4):3988–3995, 2025. doi: 10.1109/LRA.2025.3544909.

Lang Xiong, Ning Liu, Ao Ren, Yuheng Bai, Haining Fang, Binyan Zhang, Zhe Jiang, Yujuan Tan, and Duo Liu. D2 prune: Sparsifying large language models via dual taylor expansion and attention distribution awareness. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(32):27171–27179, March 2026. ISSN 2159-5399. doi: 10.1609/aaai.v40i32.39932. URL http://dx.doi.org/10.1609/aaai.v40i32.39932.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. VLA-Cache: Efficient vision-language-action manipulation via adaptive token caching. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 164448–164473. Curran Associates, Inc., 2025. doi: 10.52202/085713-5484. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/f062da1973ac9ac61fc6d44dd7fa309f-Paper-Conference.pdf.

Yantai Yang, Yuhao Wang, Zichen Wen, Luo Zhongwei, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. EfficientVLA: Training-free acceleration and compression for vision-language-action models. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 40891–40914. Curran Associates, Inc., 2025. doi: 10.52202/085713-1365. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/3a2ef31a1e45908901adc0ca853a8faf-Paper-Conference.pdf.

Fengnian Zhang, Tao Huang, Siyu Xu, Zhong Jin, and Chang Xu. Revisiting parameter redundancy in vision-language-action models: Insights from VLM-to-VLA adaptation, 2026. URL https://arxiv.org/abs/2606.31382.

A Deployment Conditions and Additional Measurements

Timing boundaries and historical references. The $\pi_{0.5}$ historical baseline is the unpruned OpenPI policy with its default `max-autotune` compilation enabled: 69.7213 ms. Its eager measurement is 201.8117 ms. The GR00T historical baselines are 52.2003 ms for Google Robot and 57.2480 ms for WidowX, using the project's inference entry with the NVIDIA benchmark DiT compilation recipe. The reference uses the project's shared pre-norm, unused-KV and RoPE handling. Table 2(b) reports their mean-latency ratios as system-level deployment improvements; the shared-backend comparisons below isolate sparse execution's contribution.

The $\pi_{0.5}$ policy boundary includes input transforms, transfers, action sampling and output transforms. GR00T times `policy._get_action`, including its observation processing and action decoding; conversion of cached observations to the policy schema occurs before timing. Sensor, simulator and network I/O, model loading, compilation and initial capture are excluded. Optimized runs share CUDA Graph replay as a backend execution facility; the real-robot deployment runtime also uses action-head capture.

Matched backend comparisons. Table 5 reports the paired controls within each optimized backend.

Table 5: **Matched backend controls.** Synchronized policy wall-time means (ms) and paired speedups. The $\pi_{0.5}$ dense control uses original weights in our adapted RealtimeVLA backend; GR00T controls use the same pruned weights under dense execution.

Model	Dense (ms)	Sparse (ms)	Speedup [95% CI]
$\pi_{0.5}$	39.91	32.85	1.215 [1.211, 1.219]
GR00T–Google	25.76	25.04	1.029 [1.005, 1.054]
GR00T–WidowX	24.97	24.33	1.026 [1.010, 1.041]

For $\pi_{0.5}$, we adapt RealtimeVLA commit `b86a942` to the original and pruned checkpoints with the same policy interface, input layout and position handling. Both use two active camera views, a 32-token text bucket, ten denoising steps and ten action tokens. The 40-task comparison measures ten repetitions per observation. Mean latency falls from 39.9139 ms for the original dense policy to 32.8540 ms for the sparse policy, a $1.2149\times$ speedup [1.2110, 1.2188]. Both modes use this matched adaptation of the backend.

The GR00T final deployment comparison fixes the pruned mathematical weights and the optimized execution backend, using three observations per task and 48 repetitions in the four-mode deployment experiment. Sparse execution reduces mean latency from 25.7646 to 25.0364 ms for Google Robot and from 24.9684 to 24.3314 ms for WidowX; speedups are $1.0291\times$ [1.0047, 1.0543] and $1.0262\times$ [1.0098, 1.0412], respectively. The intervals use 10,000 bootstrap resamples of task clusters. These results isolate sparse execution within a shared optimized backend. The operator experiments below separately evaluate fusion, including a direct comparison with the previous native sparse wrapper.

Compression scope and architecture-dependent gains. The saved $\pi_{0.5}$ projection inventory contains 1,981,808,640 language weights and 311,427,072 action-expert weights. Exact 2:4 pruning removes 990,904,320 and 155,713,536 weights, respectively, so the language backbone contributes 86.4% of the 1,146,617,856 removed weights. This percentage describes the language backbone's contribution to parameter reduction in the evaluated pruning configuration. The packed language MLP subset contains 54 matrices with 1,811,939,328 weights before packing; other pruned matrices retain dense storage and execution when sparse kernels are unprofitable. GR00T instead packs 32 gate and up matrices across 16 language layers. These inventories characterize packed weight coverage; the timing experiments measure the resulting deployment gains.

For a serial latency decomposition, let f be the fraction of dense-policy time in the projections selected for sparse execution and s their local speedup. With the remaining work fixed, the ideal policy speedup is $1/[(1-f)+f/s]$ before additional sparse overhead. Thus the packed parameter share controls weight-storage savings, whereas f and s control latency savings. This accounts for architecture-dependent benefits: backbone size, the selected projections, and dense-kernel efficiency

determine the work available for acceleration. Our experiments demonstrate compressible LLM capacity on the evaluated tasks.

A.1 Full-Sparse Execution under Official Compilation

Execution scope. The full-sparse columns in Table 2(a) retain the checkpoints and masks used for its selective eager columns, including GLUESTICK's rank-200 correction. Full-sparse execution applies standard PyTorch CUTLASS 2:4 kernels to all 252 pruned projections in $\pi_{0.5}$ and all 112 pruned language projections in GR00T, without our shape-aware selection or fused sparse operators. Unpruned modules remain unchanged. The SparseVLA row uses our pruning algorithm with this same full-sparse execution path. Each row compares against the original unpruned Dense model measured in the same run on one RTX 4090 (24 GB memory version), with batch size one and BF16.

Compilation settings. Dense and sparse modes use identical official compilation settings: OpenPI's `sample_actions` with `max-autotune` and ten denoising steps for $\pi_{0.5}$; NVIDIA's action-head DiT forward compilation with `max-autotune` and four denoising steps for GR00T. Manual CUDA Graph wrappers are disabled; compiler-managed graph optimizations remain enabled.

Timing protocol. We replay one cached simulation observation per task across all 40 LIBERO, six Google and seven WidowX tasks. Each mode receives eight warmups and 20 timed repetitions per task, with balanced execution order and matched inputs, seeds and sampling noise. Speedup is the ratio of Dense to sparse complete-policy CUDA-event medians. Loading, compilation, warmup, profiling, random-seed setup, and simulator and network I/O are excluded. No competing GPU process is detected during the accepted timing runs.

A.2 Operator Removal Ablations

We keep the selected projections, 2:4 weights and down-projection choice fixed. Six controls distinguish (A) separate standard sparse gate and up GEMMs; (B) their concatenation; (C) the configuration corresponding to the fused kernel, with a standard unfused epilogue; (D) paired gate and up rows with a separate pointwise operation; (E) epilogue fusion with the same token padding; and (F) the deployed fusion without that padding. For GR00T, A and B invoke the PyTorch sparse API. For $\pi_{0.5}$, the current backend retains FP32 accumulators through GELU and gating, so A and B use a standard CUTLASS FP32 epilogue rather than changing to the BF16-output PyTorch API. The final hidden tensor is BF16 in both cases. GR00T preserves its BF16 projection and SiLU rounding points. D uses a common standalone pointwise control, whereas the actual old GR00T wrapper has an optimized tiled pointwise transpose.

Table 6: **Controlled sparse-operator ablation.** Mean ms per MLP block (gate and up projections, activation, product, and down projection); $\pi_{0.5}$ excludes RMSNorm and residual addition. A–F use controlled library baselines; the existing GR00T native wrapper is evaluated separately below.

Model	Tokens	A	B	C	D	E	F
GR00T–Google	90	.11288	.08916	.08912	.08918	.07615	.07468
	95	.11426	.09036	.09009	.09020	.07684	.07565
GR00T–WidowX	74	.11040	.08631	.08610	.08613	.07511	.07345
	75	.11107	.08713	.08701	.08690	.07588	.07405
	77	.11135	.08718	.08726	.08709	.07601	.07456
	79	.11152	.08733	.08740	.08706	.07590	.07445
$\pi_{0.5}$	544	.88473	1.01467	.88655	.88670	.53708	.53714

Block timings in Table 6 use three real weight layers per shape, two rotations per captured call, and all 720 orders of the six controls. CUDA-event spans are divided by six; no timing samples are removed. Separate numerical checks at three input scales give bitwise-equal block outputs. Epilogue fusion provides the largest $\pi_{0.5}$ block-level reduction, from .88670 to .53708 ms. The intermediate controls show why the complete execution plan matters: concatenation alone increases

its latency from .88473 to 1.01467 ms, while timings for C and D remain similar with the paired layout. Because 544 tokens are already aligned, its E and F rows provide a duplicate-configuration control.

Table 7: **Full-policy removal experiment on $\pi_{0.5}$.** Mean ms with six graph banks sharing the same weights; all wall-clock samples are retained. The interval for A/F uses task-cluster bootstrap.

Model	A	B	C	D	E	F	A/F [95% CI]
$\pi_{0.5}$	38.31	40.70	38.31	38.30	32.35	32.34	1.185 [1.184, 1.186]

The full-policy experiment in Table 7 covers 40 $\pi_{0.5}$ observations, with 24 balanced six-mode repetitions per observation. All 4,800 non-self action comparisons against F are bitwise equal. GPU watchdogs detect no external process on the measured GPUs.

MLP comparison with the existing native wrapper. The previous GR00T native wrapper already combines gate and up projections and uses an optimized tiled SiLU–product–transpose with a dense down projection. A separate direct call of this original class method versus the deployed fusion measures the incremental benefit over that implementation. Across Google inputs of 90–95 tokens and WidowX inputs of 74, 75, 77, or 79 tokens, the MLP speedup is 1.024–$1.051\times$.

B OBS Mask Comparison Protocol

For Figure 2(b,c), we compare two matched $\pi_{0.5}$ runs using OBS under H_0 and H_J. Both start from the same dense checkpoint and use observations from 40 LIBERO tasks, correlated OBS scoring, an output-fidelity trust budget $\tau = 0.05$, and fixed-mask reconstruction blend $\beta = 0.75$. STEP is disabled ($\gamma = 0$) in both; the transition-curvature weight is $\lambda = 0$ and $\lambda = 0.1$, respectively. The zero-λ endpoint retains the same trust-selection and reconstruction procedure. Each run follows the same sequential calibration and pruning schedule, using its corresponding curvature and compensated weights.

We read the resulting checkpoints and verify exact 2:4 sparsity in all 252 attention and MLP projections. A group disagrees when its retained pair differs between the two masks. Panel (b) reports this fraction per projection and layer; each row mean averages its 18 layers equally. Panel (c) displays one contiguous 128-weight window from the layer-14 action-expert o-projection, reshaped for display. Among non-overlapping windows with changed groups in that projection, we select the first with the median positive number of disagreements. This illustrates the mask comparison; panel (a) separately compares the complete SparseVLA policy with H_0-only pruning in the illustrative rollout.

C Calibration Data and Pruning Hyperparameters

Calibration data. All calibration trajectories are collected using the corresponding dense policies. For $\pi_{0.5}$, we use one trajectory from each of the 40 LIBERO tasks, retaining at most 16 observations per task for a total of 639 calibration observations. For GR00T, we use six tasks per embodiment (Google Robot and WidowX), with four trajectories per task.

Table 8 summarizes the default pruning configuration. The symbols follow the selection and reconstruction objectives in Equations 9 and 11. Component ablations override the settings explicitly identified in their respective protocols.

Table 8: **Pruning hyperparameters.** Default values and their roles.

Parameter	Value	Meaning
Transition weight λ	0.1	Weights the stabilized transition curvature in $H_J = H_0 + \lambda H_\Delta^{\mathrm{stab}}$.
STEP weight γ	0.5	Weights the scale-matched STEP penalty in candidate-pattern selection.
Trust budget τ	0.05	Admits candidates within this fraction of the local static OBS cost range above its minimum.
Reconstruction blend β	0.75	Weights the joint-curvature solution in $(1-\beta)W_0^* + \beta W_J^*$, with the same mask at both endpoints.
Off-diagonal factor κ	1.0	Scales off-diagonal entries of the transition curvature; 1 retains them fully.
Trace scale cap	50	Maximum multiplier when matching the transition curvature's trace to the static curvature's trace.
Diagonal ratio cap	0.25	Limits the λ-weighted transition contribution relative to the static curvature at each diagonal coordinate.
Spectral ratio cap	0.25	Target upper limit on the estimated largest eigenvalue of the λ-weighted transition curvature normalized by the static diagonal.
Damping coefficient	0.01	Initial diagonal loading as a fraction of the mean curvature diagonal before factorization.
OBS update gain	1.0	Applies the full sequential OBS compensation update.
Processing block size	128	Number of input columns processed per OBS block; each sparse group still contains four weights.

Fixed choices. We use exact 2:4 sparsity (50% of each pruned matrix), correlated OBS pattern costs, and trust-constrained selection. Transition curvature uses trace matching. STEP combines input- and output-channel changes, averages them over transitions, and matches its mean magnitude within each four-weight group to the element-wise OBS reference scores before applying γ. The OBS term in the selection objective has coefficient 1.

Interpretation of the budgets. The trust threshold is $C_{H_0}^{\min} + 0.05(C_{H_0}^{\max} - C_{H_0}^{\min})$, with extrema over the six legal removal pairs in the current row and group. It constrains local candidate costs, not final model-output error or task success. The reconstruction setting gives $0.25W_0^* + 0.75W_J^*$. The trace scale cap bounds a multiplier, whereas the diagonal and spectral caps control relative curvature contributions. Spectral scaling uses a largest-eigenvalue estimate from 16 power iterations. Damping is relative to curvature scale, and numerical fallback may increase diagonal loading.

D Real-Robot Tasks and Additional Sequences

We describe the four tasks evaluated in Table 3 and illustrated in Figure 4, together with two additional conveyor-to-plate tasks. Figure 6 shows the additional toast and chocolate transfers in the FR3 setup.

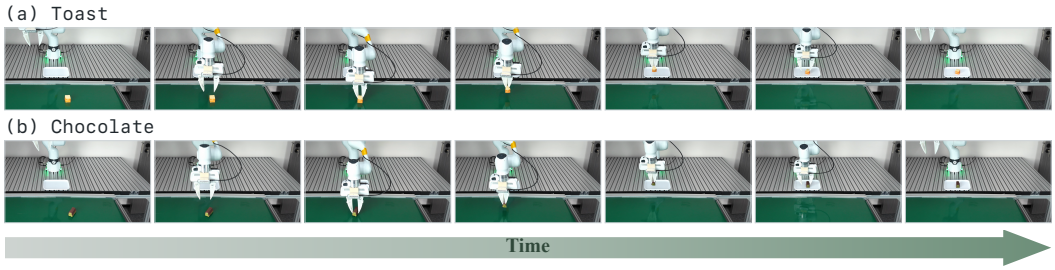


Figure 6: **Additional real-robot task sequences.** (a) Toast transfer and (b) chocolate transfer from the conveyor belt to the white plate. Each row shows seven frames in temporal order from left to right, covering approach, grasping, transport, placement, and release.

Phone placement. The robot picks up the phone from its initial position on the table and places it in the phone stand behind that position. It must establish a grasp, lift and transport the phone, align it with the stand, and release it so that the stand supports the phone after the gripper withdraws.

Pen insertion. The robot grasps the pen from the table, lifts it, and moves it toward the pen holder. It aligns the pen with the holder's opening, inserts it, and releases its grasp, leaving the pen inside the holder. This sequence combines object pickup with alignment for insertion.

Three-bowl stacking. Three bowls initially occupy separate positions on the table. The robot performs two successive transfers to combine them into a single nested stack: it first brings two bowls together, then combines the resulting stack with the remaining bowl. The final goal is to leave all three bowls stacked together after release.

Pineapple-bun transfer. The robot grasps the pineapple bun from the conveyor belt, lifts it clear of the belt, and carries it to the white plate behind the belt. It lowers the bun onto the plate and opens the gripper, leaving the bun on the plate. This is the conveyor-transfer task included in Table 3.

Toast transfer. The robot picks up the toast from the conveyor belt and places it on the white plate. Figure 6(a) shows the gripper approaching the toast, closing around it, lifting and carrying it over the plate, and releasing it before withdrawing.

Chocolate transfer. The robot picks up the chocolate from the conveyor belt and transfers it to the white plate, following the same pickup-and-placement objective with a different object. Figure 6(b) shows the approach, grasp, transport, and release, ending with the chocolate resting on the plate.