AC^2-VLA: Action-Context-Aware Adaptive Computation in Vision-Language-Action Models for Efficient Robotic Manipulation

Wenda Yu[1], **Tianshi Wang**[1*], **Fengling Li**[2], **Jingjing Li**[3], **Lei Zhu**[1*]
[1]Tongji University
[2]University of Technology Sydney
[3]University of Electronic Science and Technology of China
yu_wenda@126.com, tswang0116@163.com, fenglingli2023@gmail.com,
lijin117@yeah.net, leizhu0608@gmail.com

Abstract

Vision-Language-Action (VLA) models have demonstrated strong performance in robotic manipulation, yet their closed-loop deployment is hindered by the high latency and compute cost of repeatedly running large vision-language backbones at every timestep. We observe that VLA inference exhibits structured redundancies across temporal, spatial, and depth dimensions, and that most existing efficiency methods ignore action context, despite its central role in embodied tasks. To address this gap, we propose Action-Context-aware Adaptive Computation for VLA models (AC^2-VLA), a unified framework that conditions computation on current visual observations, language instructions, and previous action states. Based on this action-centric context, AC^2-VLA adaptively performs cognition reuse across timesteps, token pruning, and selective execution of model components within a unified mechanism. To train the adaptive policy, we introduce an action-guided self-distillation scheme that preserves the behavior of the dense VLA policy while enabling structured sparsification that transfers across tasks and settings. Extensive experiments on robotic manipulation benchmarks show that AC^2-VLA achieves up to a 1.79× speedup while reducing FLOPs to 29.4% of the dense baseline, with comparable task success.

1 Introduction

Recent progress in vision-language foundation models and large-scale robot datasets such as Open X-Embodiment [O'Neill *et al.*, 2023] have accelerated the development of generalist Vision-Language-Action (VLA) models. In closed-loop embodied tasks, these models must deliver low-latency decisions with stable closed-loop control over long-horizon tasks. Representative methods such as RT-2 [Brohan *et al.*, 2023] and OpenVLA [Kim *et al.*, 2024] demonstrate that large multimodal backbones can follow language instructions and generalize to diverse tasks. More recent policies such as CogACT [Li *et al.*, 2024a] further improve control by generating expressive action trajectories via diffusion-based modeling. However, deploying these models remains challenging because inference repeatedly executes a computationally expensive vision-language backbone at every control step, resulting in high latency and compute cost, reducing control frequency, and compromising real-time responsiveness in dynamic environments.

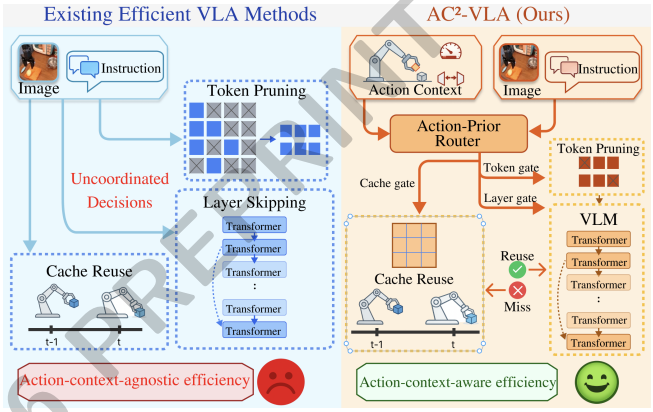

Figure 1: Comparison of efficient VLA computation strategies. Existing methods typically apply cache reuse, token pruning, or layer skipping based on visual or heuristic cues in an uncoordinated manner, resulting in action-context-agnostic efficiency. In contrast, AC^2-VLA leverages action context to jointly gate cache reuse, token pruning, and layer skipping for action-context-aware efficiency.

To mitigate these deployment challenges, recent work has explored various efficiency mechanisms for VLA models. *Static compression methods* such as pruning [Yang *et al.*, 2025] and quantization [Fang *et al.*, 2025] reduce model size but cannot adapt to changing task complexity. *Dynamic computation techniques*, including token pruning [Tan *et al.*, 2025] and layer skipping [Zhang *et al.*, 2025], adjust compute online, while caching approaches such as VLA-Cache [Xu *et al.*, 2025] exploit temporal redundancy by reusing features across adjacent timesteps. Despite these advances, most existing methods make compute allocation decisions primarily based on visual cues, which can be suboptimal for robot manipulation. In embodied tasks, visual complexity does

*Corresponding author.

not necessarily correlate with control difficulty: visually simple scenes may require full-capacity reasoning for precise interactions, while visually complex transit phases may allow more aggressive pruning.

Based on this insight, we present Action-Context-aware Adaptive Computation for VLA models (AC^2-VLA), as illustrated in Fig. 1. AC^2-VLA dynamically allocates computation along the temporal, spatial, and depth dimensions, guided by the action-centric context that is directly relevant to embodied tasks. Specifically, we introduce a lightweight action-prior router that conditions on the previous action state together with multimodal embeddings, and predicts a unified sparsity strategy for the current timestep. The router orchestrates three complementary mechanisms: *(i) cognition caching*, which reuses backbone features across adjacent timesteps when the action context suggests stable state transitions; *(ii) action-context-aware token pruning*, which removes visual tokens that are irrelevant to the current manipulation stage; and *(iii) conditional layer skipping*, which bypasses redundant transformer blocks when the action context indicates lower reasoning demand. To preserve the robustness of the original dense policy under structured sparsification, we train the router using an action-guided self-distillation scheme that encourages consistent action predictions while enabling adaptive computation. Results show that AC^2-VLA significantly reduces inference cost while maintaining strong manipulation success rates, highlighting the effectiveness of action-guided adaptive computation for efficient closed-loop control. In summary, our contributions are as follows:

- We identify that computation redundancy in VLA models aligns more with action context than visual cues, and propose action-context-aware adaptive computation for efficient robotic manipulation.
- We propose AC^2-VLA with an action-prior router that adaptively coordinates cognition caching, token pruning, and layer skipping, supported by action-guided self-distillation for robust sparsification.
- Experiments on robotic manipulation benchmarks show substantial latency and FLOPs reductions with minimal performance degradation, and confirm consistent gains over baselines with extensive ablations.

2 Related Work

2.1 Vision-Language-Action Models

The rapid progress of VLMs, together with large-scale robot data collections such as Open X-Embodiment [O'Neill *et al.*, 2023], has accelerated the emergence of generalist embodied policies that unify perception, instruction following, and action prediction. Among modern VLA systems, the RT series, such as RT-1 and RT-2, demonstrates that token-based autoregressive decoding can scale to broad task sets via large multimodal backbones [Brohan *et al.*, 2022; Brohan *et al.*, 2023]. Building on similar foundations, OpenVLA [Kim *et al.*, 2024] further systematizes training and evaluation for vision-language-action modeling at scale. In parallel, diffusion- and flow-based policies have become a compelling alternative for continuous control, where action generation is formulated as conditional denoising or flow matching and sampled as coherent trajectories, such as CogACT [Li *et al.*, 2024a] and the $\pi_0/\pi_{0.5}$ line [Black *et al.*, 2024; Black *et al.*, 2025]. Despite their strong generalization and stable closed-loop behaviors, these models share a common deployment bottleneck: regardless of paradigm, VLA inference is dominated by repeatedly executing a large multimodal backbone, leading to high latency in real-time control.

2.2 Efficient VLA Strategies

Prior work on efficient VLA strategies broadly falls into four categories: lightweight model design, dynamic routing and conditional execution, compression via pruning and quantization, and temporal reuse through caching or compute switching. Lightweight designs reduce per-step cost by scaling down backbones or streamlining training and inference, such as TinyVLA [Wen *et al.*, 2024], SmolVLA [Shukor *et al.*, 2025], and FLOWER [Reuss *et al.*, 2025]. Dynamic routing methods reduce computation by activating only a subset of the model, such as MoLe-VLA [Zhang *et al.*, 2025] and instruction-driven routing with structured sparsification in CogVLA [Li *et al.*, 2025a]. Compression-oriented approaches aim to remove redundant tokens and layers or co-design pruning with quantization, such as LightVLA [Jiang *et al.*, 2025], FlashVLA [Tan *et al.*, 2025], and SQAP-VLA [Fang *et al.*, 2025]. Temporal reuse exploits redundancy across adjacent control steps by reusing cognition or switching computation modes, such as VLA-Cache [Xu *et al.*, 2025], SP-VLA [Li *et al.*, 2025b], and VOTE [Lin *et al.*, 2025]. However, existing methods often target only one redundancy axis or rely on static heuristics without modeling action context, limiting robustness in closed-loop control. In contrast, AC^2-VLA uses action context for routing and unifies temporal reuse, spatial sparsification, and depth-wise conditional execution within an action-prior router.

3 Method

3.1 Overview

Given a visual observation x_t and a language instruction u, a VLA model predicts an action chunk $\mathbf{a}_{t:t+H}$ with horizon H. We consider a generic VLA pipeline that factorizes action generation into a multimodal backbone and an action head:

$$\mathbf{z}_t = f_{\mathrm{VLM}}(x_t, u), \qquad \mathbf{a}_{t:t+H} \sim p_\phi(\mathbf{a} \mid \mathbf{z}_t). \tag{1}$$

Here, p_ϕ can be instantiated as an autoregressive decoder or a diffusion/flow-based trajectory generator, depending on the underlying VLA policy.

In real-time deployment, inference is bottlenecked by repeatedly executing the VLM backbone at every control step. We observe structured computation redundancies along three complementary axes:

(*i*) *Temporal redundancy*: backbone representations can be reused across adjacent timesteps;

(*ii*) *Spatial redundancy*: only a subset of vision tokens is necessary for action prediction;

(*iii*) *Depth redundancy*: executing fewer backbone layers often suffices with minimal performance loss.

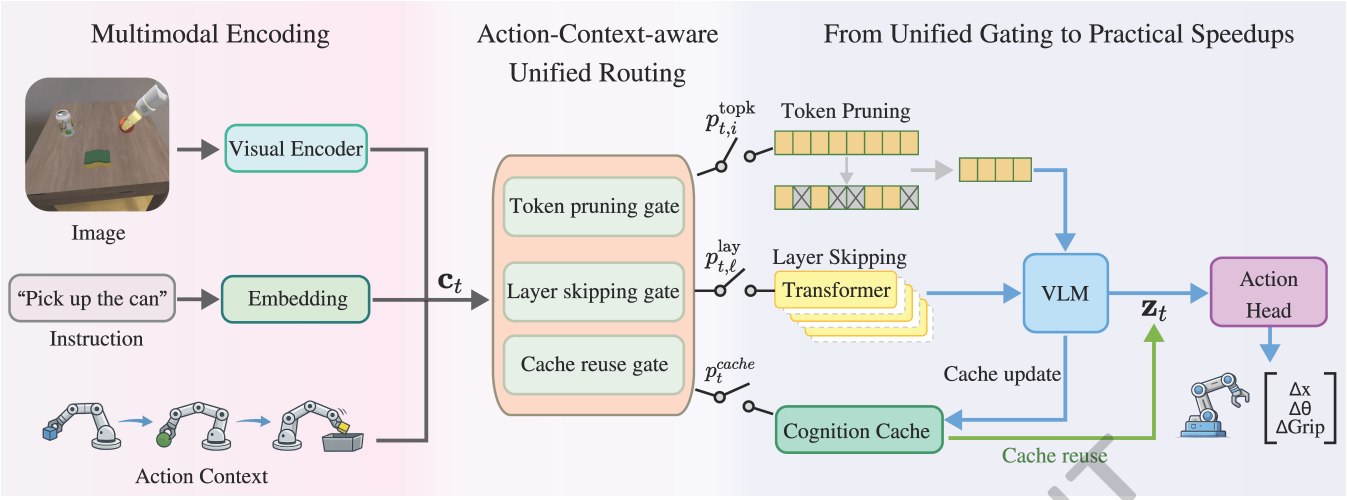

Figure 2: Overview of the proposed AC^2-VLA. At each timestep, the model builds an action-prior condition $\mathbf{c}_t$ from the current observation, instruction, and action context, and uses a unified router to generate token pruning, layer skipping, and cache reuse gates, enabling efficient computation and low-latency control.

To exploit these redundancies within a unified mechanism, AC^2-VLA introduces an action-prior router, as shown in Fig. 2. At each timestep, conditioned on the action-centric context $\mathbf{c}_t$, an action-prior router generates a set of computation gates for temporal reuse, spatial token selection, and depth-wise conditional execution:

$$\begin{aligned} \mathbf{p}_t &= [p_t^{\text{cache}};\ \mathbf{p}_t^{\text{topk}};\ \mathbf{p}_t^{\text{lay}}], \\ p_t^{\text{cache}} \in [0,1], &\quad \mathbf{p}_t^{\text{topk}} \in [0,1]^{N_v}, \quad \mathbf{p}_t^{\text{lay}} \in [0,1]^{L}, \end{aligned} \tag{2}$$

where N_v and L denote the numbers of vision tokens and transformer layers, respectively. The cache gate p_t^{cache} determines whether to reuse cached backbone representations, while $\mathbf{p}_t^{\text{topk}}$ and $\mathbf{p}_t^{\text{lay}}$ control token pruning and conditional layer execution. We train the router via teacher-student distillation with lightweight regularization to preserve dense-policy behavior under structured sparsification.

Overall, AC^2-VLA jointly optimizes temporal reuse, token selection, and conditional layer execution, enabling efficient inference for closed-loop robotic manipulation.

3.2 Action-Context-aware Unified Routing

AC^2-VLA is driven by an action-prior condition vector $\mathbf{c}_t$ that explicitly encodes the robot's action context. In closed-loop control, the next action distribution is strongly shaped by the ongoing motion state, making the previous action $\mathbf{a}_{t-1}$ a natural and inexpensive prior for allocating computation. We therefore use $\mathbf{a}_{t-1}$, parameterized consistently with the action head, as the primary routing signal. When no previous action is available at the first step, we set $\mathbf{a}_{t-1} = \mathbf{0}$ and rely on lightweight visual and instruction summaries to generate initial gates.

Let $\mathbf{V}_t \in \mathbb{R}^{N_v \times d_v}$ denote per-token vision features, and we summarize them with a mean-max mixture:

$$\mathbf{s}_t^v = \tfrac{1}{2}(\text{MeanPool}(\mathbf{V}_t) + \text{MaxPool}(\mathbf{V}_t))\,. \tag{3}$$

For language, we avoid an additional full forward by pooling embedded instruction tokens $\mathbf{E}_t \in \mathbb{R}^{T \times d}$:

$$\mathbf{s}_t^u = \tfrac{1}{2}(\mathbf{E}_t[\ell_t] + \text{MeanPool}(\mathbf{E}_t))\,, \tag{4}$$

where ℓ_t denotes the last valid token index under the attention mask when available, otherwise we use mean pooling.

We embed the action-head step index τ_t with a sinusoidal encoder $\mathbf{e}(\tau_t)$, which captures the internal generation progress of the action head. When reuse is enabled, we additionally include a cache-state cue $\mathbf{s}_t^c$ encoding the quantized action-delta proxy used for cache keying, together with compact cache statistics and an availability probe. All inputs are projected to a shared hidden size and fused by an MLP:

$$\begin{aligned} \mathbf{c}_t = f_{\text{fuse}}(&\psi_a(\mathbf{a}_{t-1}),\ \psi_v(\mathbf{s}_t^v),\ \psi_u(\mathbf{s}_t^u), \\ &\psi_\tau(\mathbf{e}(\tau_t)),\ \psi_c(\mathbf{s}_t^c)). \end{aligned} \tag{5}$$

In implementation, vision tokens and pooled summaries are detached before entering the router to prevent gradients from flowing into heavyweight backbone components through the routing pathway.

Given $\mathbf{c}_t$, the router predicts three gate families: a reuse probability p_t^{cache}, token keep scores $\mathbf{p}_t^{\text{topk}}$, and layer execution gates $\mathbf{p}_t^{\text{lay}}$. We detail each gate in the following.

Cache reuse gate. We predict a scalar reuse probability:

$$p_t^{\text{cache}} = \sigma\Big(\frac{\mathbf{w}^\top \mathbf{c}_t + b}{T_{\text{cache}}}\Big), \tag{6}$$

where T_{cache} controls gate sharpness. A high p_t^{cache} indicates a reuse request, while an actual reuse occurs only when the cache lookup succeeds.

Token pruning gate. For each vision token $\mathbf{v}_{t,i}$, we predict its keep score via action-conditioned matching:

$$p_{t,i}^{\text{topk}} = \sigma(\langle W_v \mathbf{v}_{t,i},\ W_c \mathbf{c}_t \rangle + \gamma\, g_{t,i}), \tag{7}$$

where $g_{t,i}$ is an optional lightweight bias such as a geometric prior derived from the current observation. During inference,

tokens are compacted by keeping the top-ranked ones according to $p_{t,i}^{\text{topk}}$.

Layer skipping gate. We predict per-layer execution probabilities:

$$p_{t,\ell}^{\text{lay}} = \sigma((W_\ell \mathbf{c}_t + \mathbf{b}_\ell)_\ell), \qquad \ell = 1, \ldots, L, \tag{8}$$

with bias initialization that favors near-dense execution early in training. At runtime, transformer blocks with low $p_{t,\ell}^{\text{lay}}$ are conditionally bypassed to reduce depth-wise computation.

3.3 From Unified Gating to Practical Speedups

We next describe how the unified gates translate into practical inference speedups in AC^2-VLA, through feature reuse across timesteps, spatial token pruning with compaction, and depth-wise conditional execution. Algorithm 1 summarizes the resulting inference-time procedure.

Cache reuse. When the router predicts a high reuse probability p_t^{cache}, we attempt to bypass the expensive multimodal backbone forward by querying a cognition cache. Specifically, we build a compact and robust cache key that captures both motion continuity and visual consistency. We first pool the vision tokens:

$$\bar{\mathbf{v}}_t = \text{MeanPool}(\mathbf{V}_t), \tag{9}$$

and form the cache key as

$$k_t = (\text{Quant}(\|\Delta \mathbf{a}_t\|),\ \text{Hash}(\bar{\mathbf{v}}_t)), \tag{10}$$

where $\text{Quant}(\|\Delta \mathbf{a}_t\|)$ is an action-delta norm proxy, and $\text{Hash}(\bar{\mathbf{v}}_t)$ is a lightweight vision hash for state matching. The robust hash normalizes $\bar{\mathbf{v}}_t$, applies a fixed random projection, and quantizes before hashing.

We distinguish a reuse request from an actual cache hit $h_t \in \{0, 1\}$. When a hit occurs, we directly reuse the cached multimodal representation $\mathbf{z}_t$ and skip the VLM backbone forward, otherwise we compute $\mathbf{z}_t = f_{\text{VLM}}(x_t, u)$ as usual. To keep cache population consistent with the router's intent, we only write back the newly computed $\mathbf{z}_t$ when reuse was requested but the lookup missed.

Token Pruning. Token gating determines which vision tokens should be retained. To obtain real wall-clock speedups beyond attention masking, we perform token pruning by physically removing pruned tokens and shortening the transformer sequence. Let $\mathbf{m}_t \in \{0, 1\}^{N_v}$ denote the keep mask. Compaction produces

$$\begin{aligned}(\tilde{\mathbf{V}}_t,\ \boldsymbol{\pi}_t) &= \text{Compact}(\mathbf{V}_t, \mathbf{m}_t),\\ \tilde{\mathbf{V}}_t &\in \mathbb{R}^{N_v' \times d_v},\end{aligned} \tag{11}$$

where $\boldsymbol{\pi}_t$ maps each kept token to its original patch index. We always keep at least one token to prevent degenerate empty sequences.

For Rotary Position Embedding (RoPE)-based backbones, naive reindexing after compaction would distort the original patch coordinates. We therefore preserve RoPE-consistent patch positions using $\boldsymbol{\pi}_t$:

$$\text{pos}_{t,j}^{\text{patch}} = 1 + \pi_{t,j}, \tag{12}$$

Algorithm 1 AC^2-VLA inference for a single control step t

Input: visual observation x_t, instruction u, previous action $\mathbf{a}_{t-1}$, step index τ_t, cache $\mathcal{C}$
Output: action chunk $\mathbf{a}_{t:t+H}$

$\mathbf{V}_t \leftarrow f_{\text{vis}}(x_t)$; $\bar{\mathbf{v}}_t \leftarrow \text{MeanPool}(\mathbf{V}_t)$
$\mathbf{s}_t^v \leftarrow \frac{1}{2}(\text{MeanPool}(\mathbf{V}_t) + \text{MaxPool}(\mathbf{V}_t))$
$\mathbf{s}_t^u \leftarrow \text{EmbedPool}(u)$
$\mathbf{c}_t \leftarrow f_{\text{fuse}}(\psi_a(\mathbf{a}_{t-1}), \psi_v(\mathbf{s}_t^v), \psi_u(\mathbf{s}_t^u), \psi_\tau(\mathbf{e}(\tau_t)), \psi_c(\mathbf{s}_t^c))$
$(p_t^{\text{cache}}, \mathbf{p}_t^{\text{topk}}, \mathbf{p}_t^{\text{lay}}) \leftarrow \mathcal{R}(\mathbf{c}_t)$
$h_t \leftarrow 0$
if REUSEREQ(p_t^{cache}) **then**
  $\Delta \mathbf{a}_t \leftarrow \text{DeltaProxy}(\mathbf{a}_{t-1})$
  $k_t \leftarrow (\text{Quant}(\|\Delta \mathbf{a}_t\|), \text{Hash}(\bar{\mathbf{v}}_t))$
  $(h_t, \mathbf{z}_t) \leftarrow \mathcal{C}.\text{Get}(k_t)$
end if
if $h_t = 0$ **then**
  $\mathbf{m}_t \leftarrow \text{TopKMask}(\mathbf{p}_t^{\text{topk}})$; $\mathbf{g}_t \leftarrow \text{BinGate}(\mathbf{p}_t^{\text{lay}})$
  $(\tilde{\mathbf{V}}_t, \boldsymbol{\pi}_t, N_v^{\text{orig}}) \leftarrow \text{Compact}(\mathbf{V}_t, \mathbf{m}_t)$
  $\text{pos}_t \leftarrow \text{RoPEAlign}(\boldsymbol{\pi}_t, N_v^{\text{orig}})$
  $\mathbf{z}_t \leftarrow f_{\text{VLM}}(x_t, u; \tilde{\mathbf{V}}_t, \text{pos}_t, \mathbf{g}_t)$
  if REUSEREQ(p_t^{cache}) **then**
    $\mathcal{C}.\text{Put}(k_t, \mathbf{z}_t)$ {write back only on request & miss}
  end if
end if
$\mathbf{a}_{t:t+H} \leftarrow p_\phi(\mathbf{a} \mid \mathbf{z}_t; \tau_t)$
return $\mathbf{a}_{t:t+H}$

and assign text positions to start after the original patch span N_v^{orig} recorded during compaction:

$$\text{pos}_{t,n}^{\text{text}} = 1 + N_v^{\text{orig}} + n, \quad n = 0, \ldots, T-2. \tag{13}$$

During training, to stabilize optimization and maintain differentiability, we adopt a soft relaxation by scaling projected patch embeddings with token keep scores:

$$\mathbf{e}'_{t,i} = p_{t,i}^{\text{topk}}\, \mathbf{e}_{t,i}. \tag{14}$$

Hard compaction is applied only at inference time for maximum speedup.

Layer Skipping. We implement conditional depth execution by wrapping each transformer block with a lightweight gating mechanism. Given the hidden state $\mathbf{h}^{(\ell)}$ at layer ℓ, the gated residual update is defined as

$$\mathbf{h}^{(\ell+1)} = \mathbf{h}^{(\ell)} + \alpha_{t,\ell}(F_\ell(\mathbf{h}^{(\ell)}) - \mathbf{h}^{(\ell)}), \tag{15}$$

where $F_\ell(\cdot)$ denotes the ℓ-th transformer block and $\alpha_{t,\ell} \in [0, 1]$ is the layer gate derived from $p_{t,\ell}^{\text{lay}}$. During training, $\alpha_{t,\ell}$ remains soft; at inference, it is binarized so that inactive samples bypass the layer entirely.

For efficient execution, active samples are dynamically grouped into a sub-batch to run $F_\ell(\cdot)$, and the results are scattered back to the full batch.

3.4 Optimization

We train the router to preserve dense-policy behavior under sparse execution with:

$$\mathcal{L} = \mathcal{L}_{distill} + \mathcal{L}_{reg} + \mathcal{L}_{temp}. \tag{16}$$

Google Robot	Method	Success Rate (↑)				
		PickCan	MoveNear	Drawer	DrawerApple	Average
Visual Matching	RT-1	85.7%	44.2%	73.0%	6.5%	52.4%
	RT-1-X	56.7%	31.7%	59.7%	21.3%	42.4%
	RT-2-X	78.7%	77.9%	25.0%	3.7%	46.3%
	Octo-Base	17.0%	4.2%	22.7%	0.0%	11.0%
	OpenVLA	18.0%	56.3%	63.0%	0.0%	34.3%
	CogACT	91.3%	85.0%	71.8%	50.9%	74.8%
	AC^2-VLA	**97.2%**	**82.7%**	**80.6%**	**46.8%**	**76.8%**
Variant Aggregation	RT-1	89.8%	50.0%	32.3%	2.6%	43.7%
	RT-1-X	49.0%	32.3%	29.4%	10.1%	30.2%
	RT-2-X	82.3%	79.2%	35.3%	20.6%	54.4%
	Octo-Base	0.6%	3.1%	1.1%	0.0%	1.2%
	OpenVLA	60.8%	67.7%	28.8%	0.0%	39.3%
	CogACT	89.6%	80.8%	28.3%	46.6%	61.3%
	AC^2-VLA	**88.7%**	**84.4%**	**28.2%**	**45.1%**	**61.6%**

Table 1: Google Robot success rates on SIMPLER under two evaluation settings. Most baseline results are reported by CogACT, and we add the AC^2-VLA row by evaluating our method under the same protocol.

Action-guided self-distillation. We use a teacher-student scheme, where the teacher runs the dense policy and the student executes routed sparse inference, including cache reuse, token pruning, and layer skipping. We distill both action outputs and cognition features:

$$\mathcal{L}_{distill} = \lambda_\epsilon \left\| \hat{\boldsymbol{\epsilon}}^{stu} - \hat{\boldsymbol{\epsilon}}^{tea} \right\|_2^2 + \lambda_z \mathcal{D}\big(\mathbf{z}_t^{stu}, \mathbf{z}_t^{tea}\big) . \quad (17)$$

where $\hat{\boldsymbol{\epsilon}}$ denotes the action prediction, and $\mathbf{z}_t$ is the backbone representation. λ_ϵ and λ_z control the relative weights, and $\mathcal{D}(\cdot, \cdot)$ is a feature-matching distance.

Regularization and temporal smoothing. We add $\mathcal{L}_{reg}$ to enforce target token/layer budgets and supervise the reuse gate, and $\mathcal{L}_{temp}$ to penalize abrupt changes in gating decisions across timesteps for stable closed-loop control.

4 Experiment

4.1 Experimental Setup

Backbones. We build AC^2-VLA on CogACT [Li *et al.*, 2024a], a diffusion-based VLA model with a Prismatic-7B vision-language backbone and a DiT-Base action head. To isolate routing effects, we freeze the pre-trained vision and language backbones and optimize only the lightweight routing modules, while keeping the action head unchanged with 8 denoising steps.

Implementation Details. All experiments are conducted on a node with NVIDIA RTX 5090 GPUs. We initialize from the CogACT-Base checkpoint [Li *et al.*, 2024a] and train on the Bridge subset of Open X-Embodiment [O'Neill *et al.*, 2023] for 3,000 steps using AdamW with batch size 48 and learning rate 1×10^{-6}. We use an action horizon of $H = 15$ with 8 diffusion steps. Unless noted otherwise, AC^2-VLA enables action distillation, sets the maximum token pruning ratio to 0.6, and uses a cache reuse threshold of 0.2.

Benchmarks. We evaluate on SIMPLER [Li *et al.*, 2024b], a high-fidelity simulation benchmark for robotic manipulation that aims to narrow the real-to-sim gap. We report results on two robot embodiments under three protocols:

- **Google Robot Visual Matching:** Tests generalization in visually matched real-world conditions on tasks including Pick Coke Can, Move Near, Open/Close Drawer, and Place Apple.
- **Google Robot Variant Aggregation:** Introduces variations in background, lighting, and distractors, providing a more challenging robustness setting.
- **WidowX Visual Matching:** Evaluates fine-grained manipulation on WidowX with tasks including Put Spoon on Towel, Put Carrot on Plate, Stack Cube, and Put Eggplant in Basket.

Following standard SIMPLER protocols, we use 3 Hz control with 513 Hz simulation for Google Robot and 5 Hz control with 500 Hz simulation for WidowX. Episodes are capped at 80 steps for Google Robot and 120 steps for WidowX to penalize inefficient or stalled behaviors.

Baselines. We compare AC^2-VLA with two groups of baselines: generalist dense VLA policies and efficiency-oriented methods.

Generalist VLA Models: We report results for RT-1 [Brohan *et al.*, 2022], RT-2-X [Brohan *et al.*, 2023], Octo [Octo Model Team *et al.*, 2024], OpenVLA [Kim *et al.*, 2024], and our backbone CogACT [Li *et al.*, 2024a] in full precision, which serve as dense upper bounds on task success.

Efficiency-Oriented Methods: We include representative acceleration approaches, including VLA-Cache [Xu *et al.*, 2025] for temporal reuse, EfficientVLA [Yang *et al.*, 2025] for static pruning, MoLe-VLA [Zhang *et al.*, 2025] for conditional layer skipping via mixture-of-layers routing, and FastV [Chen *et al.*, 2024] as a lightweight pruning baseline. These comparisons characterize the trade-off between compute efficiency, measured by FLOPs and latency, and manipulation success across diverse tasks.

4.2 Comparison with State-of-the-Art

We compare AC^2-VLA on SIMPLER, reporting task success in Tables 1 and 2 and the speed–accuracy trade-off in Table 3.

WidowX Robot	Method	Success Rate (↑)				
		Put Spoon on Towel	Put Carrot on Plate	Stack Cube	Put Eggplant in Basket	Average
SIMPLER Visual Matching	RT-1-X	0.0%	4.2%	0.0%	0.0%	1.1%
	Octo-Base	15.8%	12.5%	0.0%	41.7%	17.5%
	Octo-Small	41.7%	8.2%	0.0%	56.7%	26.7%
	OpenVLA	4.2%	0.0%	0.0%	12.5%	4.2%
	CogACT	71.7%	50.8%	15.0%	67.5%	51.3%
	AC^2-VLA	**71.2%**	**58.0%**	**14.8%**	**74.0%**	**54.5%**

Table 2: WidowX Visual Matching success rates on SIMPLER. Most baseline results are reported by CogACT, and we add the AC^2-VLA row by evaluating our method under the same protocol.

Setting	Method	Success Rate (↑)					Speed-up (↑)	FLOPs (↓)
		PickCan	MoveNear	Drawer	DrawerApple	Average		
Visual Matching	CogACT	91.3%	85.0%	71.8%	50.9%	74.8%	1.00×	100.0%
	VLA-Cache	92.0%	83.3%	70.5%	51.6%	74.4%	1.36×	80.1%
	EfficientVLA	95.3%	83.3%	70.3%	56.5%	76.4%	1.59×	45.1%
	FastV	92.6%	81.4%	69.8%	52.4%	74.1%	1.21×	42.0%
	MoLe-VLA	86.4%	80.2%	70.6%	50.4%	71.9%	1.53×	47.4%
	AC^2-VLA	**97.2%**	**82.7%**	**80.6%**	**46.8%**	**76.8%**	**1.79×**	**29.4%**
Variant Aggregation	CogACT	89.6%	80.8%	28.3%	46.6%	61.3%	1.00×	100.0%
	VLA-Cache	91.7%	79.3%	32.5%	45.8%	62.3%	1.37×	82.6%
	EfficientVLA	94.8%	77.6%	28.4%	51.9%	63.2%	1.57×	45.1%
	FastV	91.4%	78.6%	27.6%	50.6%	62.1%	1.19×	42.0%
	MoLe-VLA	89.2%	79.5%	29.9%	46.2%	61.2%	1.49×	46.3%
	AC^2-VLA	**88.7%**	**84.4%**	**28.2%**	**45.1%**	**61.6%**	**1.67×**	**34.7%**

Table 3: Comparison with efficiency-oriented VLA methods on SIMPLER across two settings.

Task Performance. AC^2-VLA achieves strong control performance across evaluation protocols. On Google Robot Visual Matching, it reaches 76.8% average success, outperforming the dense CogACT baseline at 74.8% and larger models such as RT-2-X at 46.3%. Gains are most pronounced on precision-critical tasks, e.g., Drawer Opening improves from 71.8% to 80.6%, suggesting that action-prior-guided sparsification helps suppress distractors and stabilizes decision making. On Variant Aggregation, AC^2-VLA matches the full-precision baseline with 61.6% versus 61.3%, while consistently surpassing RT-1 and OpenVLA.

Efficiency and Computational Cost. AC^2-VLA substantially reduces the inference cost of CogACT. As shown in Table 3, it uses 29.4% of the original FLOPs, yielding a 1.79× wall-clock speedup. Notably, this acceleration does not compromise performance and even improves success over dense CogACT, indicating that the removed computation largely corresponds to redundant or distracting features for closed-loop control.

4.3 Ablation Study

We ablate AC^2-VLA on SIMPLER Google Robot Visual Matching to validate key design choices. We focus on the three efficiency axes controlled by the router, namely token pruning, layer skipping, and cognition reuse, and analyze their interaction under comparable budgets. Ablation results are summarized in Table 4. We observe that each component provides complementary benefits, and the full model achieves the best speed-accuracy trade-off when all three axes

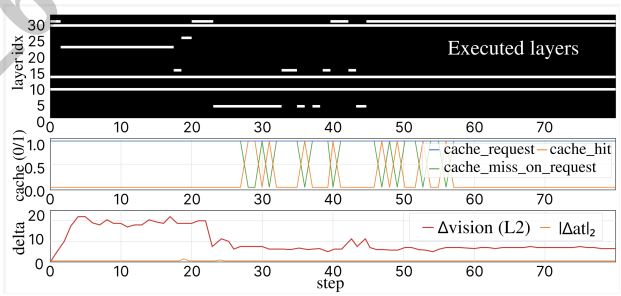

Figure 3: Adaptive layer execution and cache reuse over time.

are jointly enabled:

- **Cache reuse.** Without cognition reuse, success drops to 70.5% and speedup decreases to 1.66×, suggesting that temporal reuse improves both efficiency and closed-loop stability. Fig. 3 illustrates adaptive layer execution and cache hit behavior over time.
- **Token pruning.** Removing token pruning reduces speedup to 1.52×, showing that spatial sparsification contributes most to acceleration. Fig. 4 visualizes the token-level routing patterns.
- **Layer routing.** Disabling layer routing drops the success rate to 67.4% under similar FLOPs, indicating that conditional depth execution helps retain high-level reasoning while reducing compute.
- **Full model.** AC^2-VLA attains 76.8% success while delivering a 1.79× speedup, demonstrating the effective-

Figure 4: Left: input observation. Right: visualization of token-level importance predicted by the action-conditioned router, highlighting regions relevant to the current manipulation stage while suppressing distractors. The highlighted regions adapt with the action context, focusing computation on interaction-critical areas.

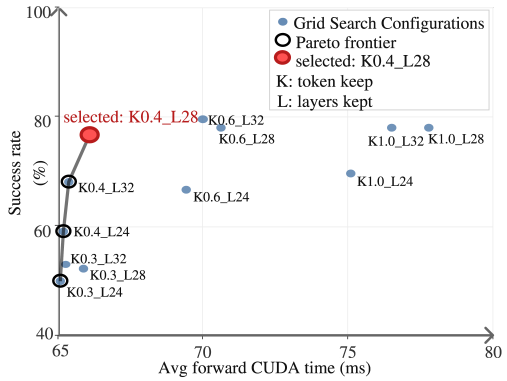

Figure 5: Pareto frontier for token pruning and layer skipping on the SIMPLER benchmark.

ness of jointly leveraging spatial, depth-wise, and temporal redundancies.

4.4 More Exploration

Beyond component ablations, we further analyze the hyper-parameter space and emergent behaviors of AC^2-VLA, focusing on the joint sparsity trade-off and the effect of cognition caching on closed-loop stability.

Token-layer sparsity and the efficiency-accuracy trade-off. We perform a grid search over the token keep ratio r_{topk} and the executed layer count N_{lay}, as shown in Fig. 5. The results exhibit a clear Pareto frontier, where $r_{topk} = 0.4$ and $N_{lay} = 28$ achieves the best trade-off, reaching 1.79× speedup with 76.8% success. This suggests that many visual tokens are dispensable, while sufficient depth remains important for reasoning over the retained tokens.

When cache reuse improves robustness beyond speed. Interestingly, cache reuse can improve robustness in addition to reducing compute. As shown in Table 5, setting the cache threshold to $\tau_{cache} = 0.2$ yields 87.1% success, outperforming the dense baseline by +12.3%. We attribute this gain to improved temporal consistency: standard per-frame inference can amplify high-frequency visual noise and induce action jitter, whereas reusing $\mathbf{z}_t$ when the action context is stable effectively smooths decision making and stabilizes control.

Sensitivity to key hyper-parameters. We next vary the token keep ratio and the maximum executed depth to examine how accuracy degrades under more aggressive sparsification.

- **Token sparsity.** Performance remains stable down to $r_{topk} = 0.4$, but collapses at $r_{topk} = 0.2$ with 33.3% success, indicating a minimum visual information requirement for manipulation.
- **Depth.** The policy remains competitive with $N_{lay} = 28$ at 77.3% success, while reducing below 24 layers causes a sharp drop, suggesting that sufficient depth is critical for complex tasks.

Configuration	Success Rate (↑)	Speed-up (↑)	FLOPs (↓)
Dense baseline	74.8%	1.00×	100.0%
Full AC^2-VLA	76.8%	1.79×	29.4%
No cache reuse, $\tau_{cache} = 1.0$	70.5%	1.66×	38.6%
Without layer routing	67.4%	1.68×	29.4%
Without token pruning	72.7%	1.52×	66.8%

Table 4: Component ablation on SIMPLER Google Robot Visual Matching. Speed-up is measured relative to the dense baseline.

Sweep	Value	Success Rate (↑)	Speed-up (↑)	FLOPs (↓)
Token keep ratio r_{topk}	0.2	33.3%	1.69×	25.8%
	0.3	56.1%	1.66×	34.7%
	0.4	68.9%	1.63×	44.1%
	0.6	78.8%	1.47×	62.5%
	0.8	81.1%	1.26×	81.0%
Kept layers N_{lay}	22	69.7%	1.51×	68.8%
	24	73.5%	1.46×	75.0%
	26	78.0%	1.41×	81.2%
	28	77.3%	1.37×	87.5%
	30	80.3%	1.33×	93.8%
Cache threshold τ_{cache}	0.00	82.6%	1.53×	64.8%
	0.05	81.1%	1.52×	65.8%
	0.10	77.3%	1.54×	63.4%
	0.20	87.1%	1.53×	63.4%
	0.30	78.8%	1.51×	66.3%
	0.40	79.5%	1.52×	64.9%

Table 5: Sensitivity sweeps on SIMPLER Google Robot Visual Matching, varying one efficiency component at a time with the other two disabled.

5 Conclusion

We present AC^2-VLA, an action-context-aware framework for efficient Vision-Language-Action inference. By introducing a unified router that allocates computation across spatial, depth, and temporal dimensions based on the robot's manipulation state, AC^2-VLA addresses the limitations of efficiency methods driven solely by visual complexity and enables adaptive closed-loop control. Experiments on the SIMPLER benchmark demonstrate a superior efficiency–accuracy trade-off, achieving a 1.79× speedup and reducing FLOPs to 29.4% of the dense baseline while improving task success. These results indicate that action-guided sparsification acts as both an efficiency mechanism and a regularizer, suppressing visual distractors and promoting temporal consistency. Overall, AC^2-VLA suggests that aligning computation with action is a more effective paradigm for embodied intelligence than static compression, and points toward adaptive inference as a key ingredient for scalable generalist robot policies.

Ethical Statement

There are no ethical issues.

Acknowledgments

This work was supported in part by the New Generation Artificial Intelligence-National Science and Technology Major Project (No. 2025ZD0123000), the Central Guidance on Local Science and Technology Development Fund of Shanghai City (No. YDZX20253100002004), the China Postdoctoral Science Foundation (No. 2025M781504), the New Cornerstone Science Foundation through the XPLORER PRIZE, and the Fundamental Research Funds for the Central Universities.

References

[Black *et al.*, 2024] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[Black *et al.*, 2025] 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, editors, *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 17–40. PMLR, 27–30 Sep 2025.

[Brohan *et al.*, 2022] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[Brohan *et al.*, 2023] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[Chen *et al.*, 2024] Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. *arXiv preprint arXiv:2403.06764*, 2024.

[Fang *et al.*, 2025] Hengyu Fang, Yijiang Liu, Yuan Du, Li Du, and Huanrui Yang. Sqap-vla: A synergistic quantization-aware pruning framework for high-performance vision-language-action models. *arXiv preprint arXiv:2509.09090*, 2025.

[Jiang *et al.*, 2025] 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. *arXiv preprint arXiv:2509.12594*, 2025.

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

[Li *et al.*, 2024a] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, Xiaofan Wang, Bei Liu, Jianlong Fu, Jianmin Bao, Dong Chen, Yuanchun Shi, Jiaolong Yang, and Baining Guo. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

[Li *et al.*, 2024b] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, 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. *arXiv preprint arXiv:2405.05941*, 2024.

[Li *et al.*, 2025a] Wei Li, Renshan Zhang, Rui Shao, Jie He, and Liqiang Nie. CogVLA: Cognition-aligned vision-language-action model via instruction-driven routing & sparsification. *arXiv preprint arXiv:2508.21046*, 2025.

[Li *et al.*, 2025b] Ye Li, Yuan Meng, Zewen Sun, Kangye Ji, Chen Tang, Jiajun Fan, Xinzhu Ma, Shutao Xia, Zhi Wang, and Wenwu Zhu. SP-VLA: A joint model scheduling and token pruning approach for VLA model acceleration. *arXiv preprint arXiv:2506.12723*, 2025.

[Lin *et al.*, 2025] Juyi Lin, Amir Taherin, Arash Akbari, Arman Akbari, Lei Lu, Guangyu Chen, Taskin Padir, Xiaomeng Yang, Weiwei Chen, Yiqian Li, Xue Lin, David Kaeli, Pu Zhao, and Yanzhi Wang. VOTE: Vision-language-action optimization with trajectory ensemble voting. *arXiv preprint arXiv:2507.05116*, 2025.

[Octo Model Team *et al.*, 2024] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

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

[Reuss *et al.*, 2025] Moritz Reuss, Hongyi Zhou, Marcel Rühle, Ömer Erdinç Yağmurlu, Fabian Otto, and Rudolf Lioutikov. FLOWER: Democratizing generalist robot policies with efficient vision-language-action flow policies. *arXiv preprint arXiv:2509.04996*, 2025.

[Shukor *et al.*, 2025] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Rémi Cadène. SmolVLA: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[Tan *et al.*, 2025] Xudong Tan, Yaoxin Yang, Peng Ye, Jialin Zheng, Bizhe Bai, Xinyi Wang, Jia Hao, and Tao Chen. Think twice, act once: Token-aware compression and action reuse for efficient inference in Vision-Language-Action models. *arXiv preprint arXiv:2505.21200*, 2025.

[Wen *et al.*, 2024] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, Yaxin Peng, Feifei Feng, and Jian Tang. TinyVLA: Towards fast, data-efficient Vision-Language-Action models for robotic manipulation. *arXiv preprint arXiv:2409.12514*, 2024.

[Xu *et al.*, 2025] Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Efficient vision-language-action manipulation via adaptive token caching. *arXiv preprint arXiv:2502.02175*, 2025.

[Yang *et al.*, 2025] 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. *arXiv preprint arXiv:2506.10100*, 2025.

[Zhang *et al.*, 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.