LEARNING TO ACCELERATE VISION-LANGUAGE-ACTION MODELS THROUGH ADAPTIVE VISUAL TOKEN CACHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) models have demonstrated remarkable generalization capabilities in robotic manipulation tasks, yet their substantial computational overhead remains a critical obstacle to real-world deployment. Improving inference efficiency is therefore essential for practical robotic applications. Existing acceleration methods often rely on heuristic or static strategies—such as rule-based token caching or pruning—that are decoupled from task objectives and fail to adapt to dynamic scene changes. In this work, we reformulate inference acceleration as a learnable policy optimization problem and propose a novel framework that integrates a dynamic, task-aware decision-making process directly into the VLA model. At its core are two lightweight, cooperative modules: a *Cached Token Selector*, which determines *which* tokens should be reused, and a *Cache Ratio Predictor*, which controls *how many* tokens to reuse. Training these modules is non-trivial due to their discrete decisions. We address this by adopting a differentiable relaxation that allows gradient-based end-to-end optimization. Extensive experiments on the LIBERO and SIMPLER benchmarks, as well as real-robot evaluations, show that our method achieves a 1.76$\times$ wall-clock inference speedup while simultaneously improving the average success rate by 1.9 percentage points (from 75.0% to 76.9%) on LIBERO and by 5.0 percentage points on real-world tasks, significantly outperforming existing baselines. This work highlights the potential of learning task-aware computational allocation policies, paving the way for VLA models that are both powerful and efficient.

1 INTRODUCTION

Large Vision-Language-Action (VLA) models have recently shown remarkable progress in unifying perception, reasoning, and control for embodied intelligence (Zitkovich et al., 2023; Driess et al., 2023; Kim et al., 2024; Li et al., 2024a; Black et al., 2024; Zhao et al., 2025; Intelligence et al., 2025; Zhou et al., 2025; Zhong et al., 2025). By coupling large-scale multimodal pretraining with autoregressive action decoding, models such as RT-2 (Zitkovich et al., 2023), PaLM-E (Driess et al., 2023), and OpenVLA (Kim et al., 2024) enable robots to interpret open-ended language commands and perform complex manipulation tasks. However, deploying such models in real-world robotic systems remains challenging due to their high computational overhead, which conflicts with the low-latency requirements of real-time control, hindering widespread adoption.

VLA inference comprises multimodal prefilling, which encodes the current observation and instruction, and action decoding, which produces executable actions. Unlike long-form language generation, closed-loop control repeatedly encodes hundreds of visual tokens at every step, while autoregressive VLAs such as OpenVLA produce only about seven action tokens. Visual prefilling is therefore a recurrent perception-side cost. Although action chunking reduces how often actions are generated, it does not eliminate the need to refresh visual representations as observations change, making visual prefilling a complementary target for accelerating responsive closed-loop control.

A major inefficiency arises from redundant computation across consecutive frames, as VLA models reprocess the entire visual input at every step, even when most background regions remain static. Existing acceleration methods attempt to mitigate this by caching or pruning visual tokens, but

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.

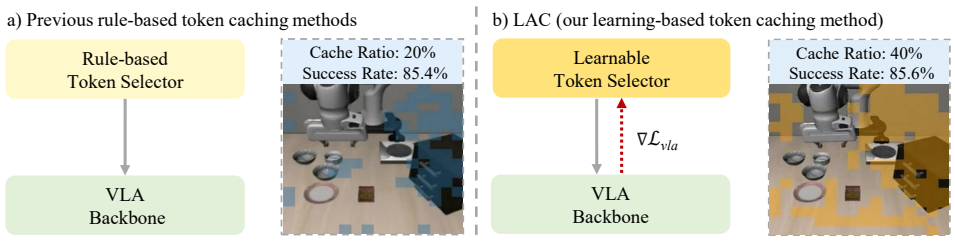


Figure 1: Our learnable caching (LAC) outperforms rule-based baselines in both efficiency and performance. **(a)** Rule-based selectors are decoupled from the task objective. They often rely on proxies, but *where the model attends is not necessarily what the task requires*. This leads to sub-optimal, task-agnostic caching (20% cache ratio). **(b)** Our selector is optimized with direct task gradients ($\nabla\mathcal{L}_{\text{VLA}}$), ensuring its decisions align with actual task requirements. This enables a more aggressive adaptive caching strategy, leading to a much higher cache ratio (40%) and greater efficiency, while simultaneously improving the success rate.

they typically rely on fixed, rule-based heuristics (Chen et al., 2024; Zhang et al., 2024b; Xu et al., 2025). As illustrated in Figure 1 (a), these rule-based selectors are fundamentally decoupled from the downstream task objective. This means their decisions are based on proxies, such as attention scores or visual saliency, which often fail to capture true task-relevance. A policy might waste computation on visually distracting but functionally irrelevant regions, or erroneously discard subtle but critical cues for the task. In essence, ***where the model attends is not necessarily what the task requires***. This fundamental misalignment leads to a suboptimal trade-off between speed and accuracy, limiting both the potential efficiency gains and the overall reliability of the system.

We argue that the key to efficient VLA inference lies not in manually designing better heuristics, but in learning a dynamic computation policy that is directly optimized for task success. We hypothesize that an optimal policy should learn to allocate computational resources based on scene motion and task requirements, reusing redundant information while recomputing only the most salient parts.

Based on this insight, we propose **LAC** (**L**earnable **A**daptive **C**aching for Vision-Language-Action models), a lightweight framework that transforms heuristic acceleration into a learnable, task-driven policy. As shown in Figure 1 (b), LAC introduces a *learnable token selector* that receives direct feedback from the final task loss ($\nabla\mathcal{L}_{vla}$) which measures the ability of VLA models. This end-to-end optimization enables the selector to make more intelligent, task-aware decisions. As a result, LAC can adopt a more aggressive caching strategy (40% cache ratio) that not only improves efficiency but also enhances task performance, achieving a higher success rate. At its core, LAC features two cooperative modules: the *Cached Token Selector*, which learns token-level dynamic saliency, and the *Cache Ratio Predictor*, which determines the overall reuse budget. To effectively assess scene dynamics, these modules leverage computationally lightweight optical flow. This approach provides a direct, pixel-level signal of motion, which is crucial for identifying which visual tokens correspond to moving objects like the robot's gripper or manipulated items. Crucially, we opt for this direct motion signal over language-based guidance, as the latter would likely necessitate a larger model and incur additional computational overhead, running counter to our goal of efficiency. Both are trained jointly through a task-driven policy learning framework, ensuring that every computation decision contributes directly to the final goal.

Before delving into the details, we summarize our contributions as follows:

- We propose **LAC**, the first learnable adaptive caching framework that transforms heuristic acceleration into an end-to-end trainable policy for VLA models.
- We design two lightweight and cooperative decision modules—the *Cached Token Selector*, which identifies which visual tokens should be recomputed or reused, and the *Cache Ratio Predictor*, which determines how many tokens to reuse based on scene-level motion entropy. Together, they enable adaptive token reuse for efficient VLA inference.

- Extensive experiments on LIBERO, SIMPLER, and real-world robotic platforms demonstrate that LAC achieves up to a 1.76× wall-clock acceleration while simultaneously improving task performance, outperforming rule-based pruning and caching baselines.

2 Related Work

Vision-Language-Action Models. Vision-Language-Action (VLA) models represent a significant advancement in robotics, building upon the multimodal reasoning capabilities of large-scale vision-language models (VLMs) (Alayrac et al., 2022; Li et al., 2023; Liu et al., 2024a; Bai et al., 2023; Dai et al., 2023; Du et al., 2022; Wang et al., 2024; Shen et al., 2024). By introducing an action modality, these models (Zitkovich et al., 2023; Kim et al., 2024; Li et al., 2024a; Team et al., 2024; Li et al., 2024b; Liu et al., 2025b; Li et al., 2025b; Liu et al., 2025a; Wang et al., 2025; Wen et al., 2023; Shi et al., 2025; Zhou et al., 2025) are able to perform end-to-end visuomotor control. The common paradigm involves adapting powerful VLM backbones, such as Llama (Touvron et al., 2023; Grattafiori et al., 2024), through fine-tuning on extensive robotics datasets like Open X-Embodiment (O'Neill et al., 2024). To generate actions, these models employ diverse strategies, from discretizing robot movements into language-like tokens (Zitkovich et al., 2023; Kim et al., 2024) to integrating specialized decoders like diffusion action transformers (Li et al., 2024a; Wen et al., 2025c;a) or flow models (Black et al., 2024; Intelligence et al., 2025). This has led to successes in complex manipulation tasks (Huang et al., 2024; Zhao et al., 2023), but the substantial computational demands of VLA models remain a critical hurdle for practical, real-time deployment in resource-constrained settings.

Acceleration of VLA Models. Prior work on accelerating VLMs often falls short for robotic applications. Strategies like token pruning (Chen et al., 2024; Zhang et al., 2024b; Cao et al., 2024; Lin et al., 2024; Xing et al., 2024) and merging (Cao et al., 2023; Hu et al., 2025; Li et al., 2024d; Shang et al., 2025) are ill-suited for robotics. They fail to exploit critical temporal redundancy across video frames. Moreover, their pruning logic can harm the spatial fidelity required for manipulation and introduces computational overhead that often outweighs the benefits for short robotic action sequences. VLA-specific approaches include architectural optimizations (Liu et al., 2024b; Wen et al., 2025b), dynamic scheduling (Yue et al., 2024; Zhang et al., 2025; Li et al., 2025a), and quantization (Park et al., 2024). Recent methods more directly target VLA inference: SP-VLA (Li et al., 2026) jointly schedules models and prunes visual tokens. High-frequency control methods such as FAST (Pertsch et al., 2025) reduce the cost of action generation through more efficient action representations or decoding (Zhang et al., 2024a; Kim et al., 2025). LAC addresses a different source of latency: the repeated processing of visual observations across control steps. The two directions are complementary rather than mutually exclusive. Decoding acceleration shortens output-side computation, while cross-frame visual caching reduces the perception-side computation that remains whenever the policy incorporates a new observation. The most related caching approach, VLA-Cache (Xu et al., 2025), reuses key-value states across timesteps through a static, rule-based policy that cannot adapt to changing scene dynamics.

In contrast, LAC learns fine-grained cross-frame token-state reuse under the downstream task objective. It neither skips entire action-generation steps nor permanently removes visual tokens; instead, it dynamically decides which states to recompute, cache, or recover as the scene evolves.

3 Method

Given a sequence of visual observations, LAC reduces redundant perception computation by selectively reusing cached visual tokens across timesteps. It augments a frozen VLA backbone with two lightweight decision modules and is trained in two stages, as summarized in Figure 2.

3.1 Overall Framework

LAC determines both which visual tokens to recompute and how many tokens to reuse. At each timestep, the Cached Token Selector assigns token-level saliency scores, while the Cache Ratio Predictor estimates the reuse budget from scene dynamics. Together, they construct a binary mask

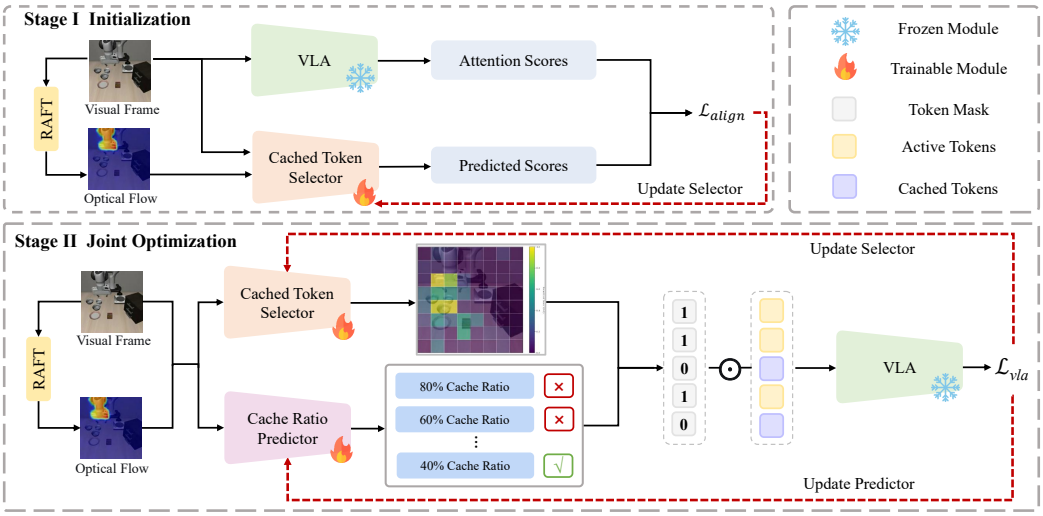


Figure 2: **Overview of the two-stage training framework for LAC. Stage I: Initialization.** The Cached Token Selector is initially pretrained to align with the VLA's internal attention map, providing a stable and informative initialization. **Stage II: Joint Optimization.** Following this, the Cached Token Selector and Cache Ratio Predictor are jointly optimized with the VLA backbone held frozen. The selector learns token-level saliency for dynamic reuse, while the predictor estimates the optimal cache ratio from a discrete set. Gradients from the task loss $\mathcal{L}_{\text{VLA}}$ flow through both modules via differentiable selection, enabling end-to-end policy learning for adaptive computation.

that routes salient tokens through the VLA and retrieves the remaining states from the preceding frame. The pre-trained VLA backbone remains frozen throughout training.

Training proceeds in two stages. Stage I initializes the Cached Token Selector by matching the expert VLA's attention distribution, providing a stable saliency prior. Stage II jointly optimizes the selector and predictor using the downstream task objective, directly aligning caching decisions with control performance. The following sections detail the modules and training objectives.

3.2 Preliminaries: KV Caching for Efficiency

Given an input sequence $\mathbf{X}$, a Transformer computes queries, keys, and values as

$$\mathbf{Q} = \mathbf{X}W_Q, \quad \mathbf{K} = \mathbf{X}W_K, \quad \mathbf{V} = \mathbf{X}W_V, \tag{1}$$

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\right)\mathbf{V}. \tag{2}$$

Autoregressive decoding appends only the new key and value states:

$$\mathbf{K}_t = [\mathbf{K}_{t-1}, \mathbf{k}_{\text{new}}], \quad \mathbf{V}_t = [\mathbf{V}_{t-1}, \mathbf{v}_{\text{new}}]. \tag{3}$$

This avoids repeatedly computing historical states within the same context. In closed-loop VLA inference, however, models typically re-encode every visual observation even though consecutive frames are highly redundant. LAC extends reuse across frames by retrieving cached visual states and recomputing only task-relevant tokens, learning *which* tokens to update and *how many* to recompute.

3.3 Dynamic Token Caching Modules

At the core of our framework are two lightweight, cooperative modules: the *Cached Token Selector* and the *Cache Ratio Predictor*. To capture scene dynamics, they leverage a motion-aware representation $V_t = [I_t; O_t]$, formed by concatenating the visual frame I_t and the optical flow O_t. We compute O_t with a lightweight RAFT-small (Teed & Deng, 2020) model, as it provides a direct and efficient motion signal, avoiding the higher computational overhead associated with using language for guidance. These modules jointly decide *which* visual tokens to recompute and *how many* to reuse

from the cache. For an input token sequence $X_t = \{x_t^{(1)}, \ldots, x_t^{(N)}\}$, they operate on V_t to produce a binary mask $M_t \in \{0,1\}^N$ that guides the per-token computation.

3.3.1 CACHED TOKEN SELECTOR

The Cached Token Selector is a function f_{sel} with parameters θ_{sel}, responsible for assessing token-level importance. Given the motion-aware input V_t, it generates a vector of saliency scores S_t:

$$S_t = f_{\text{sel}}(V_t; \theta_{\text{sel}}), \tag{4}$$

where $S_t = \{s_t^{(1)}, \ldots, s_t^{(N)}\}$ and $s_t^{(i)}$ is a continuous value in the range $[0,1]$. A higher score $s_t^{(i)}$ indicates that token $x_t^{(i)}$'s information is critical for the current timestep—perhaps due to manipulator movement or object interaction—and should accordingly be recomputed. Conversely, tokens with lower scores are identified as candidates for caching. To ensure minimal computational overhead, f_{sel} is implemented as a small and efficient convolutional neural network (CNN).

3.3.2 CACHE RATIO PREDICTOR

While the selector provides a relative ranking, the Cache Ratio Predictor determines the overall computational budget. This module, $f_{\text{pred}}(\cdot; \theta_{\text{pred}})$, also assesses global scene dynamics using the input V_t to select an appropriate cache ratio from a discrete set $\mathcal{R} = \{r_1, \ldots, r_C\}$ by outputing a vector of logits L_t over the set $\mathcal{R}$:

$$L_t = f_{\text{pred}}(V_t; \theta_{\text{pred}}), \tag{5}$$

where $L_t = \{l_t^{(1)}, \ldots, l_t^{(C)}\}$. Each logit $l_t^{(j)}$ corresponds to the model's confidence in applying cache ratio r_j. During inference, the ratio with the highest logit is selected. In a static scene, the predictor is trained to select a high ratio to maximize efficiency, while in a dynamic scene, it learns to select a lower ratio to maintain high task performance.

3.4 TWO-STAGE TRAINING PROCEDURE

Learning a discrete caching policy from scratch is unstable under sparse supervision. We address this cold-start problem with a two-stage training paradigm: we first initialize the policy by distilling expert knowledge, then fine-tune it on task objectives while keeping the pretrained VLA frozen.

Stage I: Initialization via Attention Alignment. This stage provides the *Cached Token Selector* with an initialization by distilling knowledge from the expert VLA. Although the VLA's attention scores are not optimal policies, they offer a useful proxy for general visual saliency. As shown in Figure 2, the selector f_{sel} is trained to mimic the VLA's attention map S_{VLA} by minimizing a MSE loss:

$$\mathcal{L}_{\text{align}} = \text{MSE}(f_{\text{sel}}(V_t; \theta_{\text{sel}}), S_{\text{VLA}}). \tag{6}$$

This warm-up provides the selector with a sensible initial policy, which is crucial for stabilizing the subsequent end-to-end optimization process.

Stage II: Joint Optimization for Task Performance. After initialization, both modules are jointly optimized to balance task accuracy and computational efficiency. To make discrete selections differentiable, we employ the Gumbel-Softmax trick with straight-through estimation (see Sec. 3.5). The total loss combines the task loss $\mathcal{L}_{\text{VLA}}$ with a regularization term encouraging higher cache ratios:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{VLA}} + \lambda \mathcal{L}_{\text{ratio}}, \tag{7}$$

where

$$\mathcal{L}_{\text{ratio}} = -\mathbb{E}_{\tilde{p}_t}[r] = -\sum_{j=1}^{C} \tilde{p}_t^{(j)} r_j. \tag{8}$$

Here, $\tilde{p}_t$ denotes the soft probabilities from Gumbel-Softmax and r_j the candidate cache ratios. The weight λ controls the efficiency–performance trade-off. Gradients are backpropagated to update both modules. This end-to-end training aligns the learned caching policy directly with the downstream control objective.

3.5 Differentiable Discrete Decision Learning

Both LAC modules make discrete decisions: the Cache Ratio Predictor applies $\arg\max$ over $\mathcal{R}$, while the Cached Token Selector uses `top-k` to construct the cache mask. Because these operations block gradients from the task loss, we combine Gumbel-Softmax (Jang et al., 2017) with the Straight-Through Estimator (STE) (Bengio et al., 2013), using discrete decisions in the forward pass and continuous surrogates during backpropagation.

For the Cache Ratio Predictor, logits l_t are converted into a soft distribution:

$$\tilde{p}_t^{(j)} = \frac{\exp((l_t^{(j)} + g_j)/\tau)}{\sum_{k=1}^{C} \exp((l_t^{(k)} + g_k)/\tau)}, \tag{9}$$

where $g_j \sim \mathrm{Gumbel}(0, 1)$ and τ is the temperature. The forward pass selects a deterministic one-hot ratio,

$$p_t = \mathrm{one_hot}\left(\arg\max_j \tilde{p}_t^{(j)}\right), \tag{10}$$

while the STE propagates gradients through $\tilde{p}_t$.

The Cached Token Selector follows the same principle: the forward pass uses the hard `top-k` mask $\mathbf{M}_t$, while the backward pass uses

$$\tilde{M}_t^{(i)} = \sigma\left(\frac{s_t^{(i)} - \theta_k}{\tau_s}\right), \tag{11}$$

where θ_k is the score threshold of the k_t-th token and τ_s is the temperature. These differentiable surrogates allow LAC to learn *what* and *how much* to cache from the downstream task objective.

3.6 Inference Procedure

Once the training process is complete, our decision modules are integrated with the frozen VLA backbone to create a highly efficient and stable inference pipeline.

Deterministic Decision-Making. For stable and consistent inference, we replace the stochastic sampling from training with deterministic $\arg\max$ operations. The *Cache Ratio Predictor* selects the cache ratio r_t corresponding to the highest logit. Subsequently, the *Cached Token Selector* generates a binary mask $\mathbf{M}_t$ by selecting the $k_t = N \cdot r_t$ tokens with the lowest importance scores.

Efficient Forward Pass. The generated mask $\mathbf{M}_t$ partitions visual tokens to enable an efficient forward pass. *Active tokens* are re-encoded to compute new Key (K) and Value (V) states, while the KV states of *cached tokens* are directly reused from the previous timestep. This merged set of KV states is then fed to the action decoder, significantly reducing redundant computation.

Stochastic Recovery Mechanism. To enhance long-horizon robustness and mitigate error accumulation from persistent caching, we employ a stochastic recovery mechanism. At each step, with a small probability p_{recover}, a fraction of "cached" tokens are randomly selected and forced to be recomputed. This low-cost refresh improves model stability with negligible efficiency loss.

4 Experiments

We evaluate LAC on three open-source Vision-Language-Action (VLA) backbones. OpenVLA (Kim et al., 2024) and $\pi_{0.5}$ (Intelligence et al., 2025) are evaluated on the four LIBERO (Liu et al., 2023) suites, while CogAct (Li et al., 2024a) is evaluated under the Visual Matching and Variant Aggregation settings of SIMPLER (Li et al., 2024c). We further evaluate a LoRA-finetuned (Hu et al., 2022) OpenVLA policy on a physical Franka manipulator across four tasks, using 25 trials per task and method, for 300 trials in total.

4.1 Results on Simulation Environments

Main Results on LIBERO. With OpenVLA, LAC achieves a 76.9% average success rate and a 1.76× speedup, outperforming SP-VLA at 74.9% and 1.35×. The advantage is most pronounced

Table 1: **Comparison on LIBERO with OpenVLA and $\pi_{0.5}$ backbones.** LAC achieves the highest average success rate and largest speedup among accelerated OpenVLA methods, while retaining 96.4% success with a 1.22× speedup on $\pi_{0.5}$. Note that reported speedups include all policy and optical-flow overheads (breakdown in Appendix C.1).

Method	LIBERO Spatial	LIBERO Object	LIBERO Goal	LIBERO Long	Average	FLOPs (%, ↓)	Speedup ↑
Baseline (OpenVLA)	84.4%	86.6%	75.6%	53.2%	75.0%	100.0	1.00×
SparseVLM	79.8%	67.0%	72.6%	39.4%	64.7%	75.5	0.62×
FastV	83.4%	84.0%	74.2%	51.6%	73.3%	71.0	0.97×
FastVLM	79.8%	77.0%	73.2%	36.6%	66.65%	86.2	1.16×
VisionZip	34.2%	47.4%	46.0%	4.6%	33.05%	81.9	1.21×
VLA-Cache	83.8%	85.8%	76.4%	52.8%	74.7%	72.7	1.63×
SP-VLA	84.4%	85.6%	75.4%	54.2%	74.9%	73.6	1.35×
Ours (LAC)	**85.6%**	**86.2%**	**76.6%**	**59.2%**	**76.9%**	**74.7**	**1.76×**
Baseline ($\pi_{0.5}$)	98.8%	98.2%	98.0%	92.4%	96.85%	100.0	1.00×
Ours (LAC)	**98.2%**	**97.8%**	**97.4%**	**92.2%**	**96.4%**	**88.1**	**1.22×**

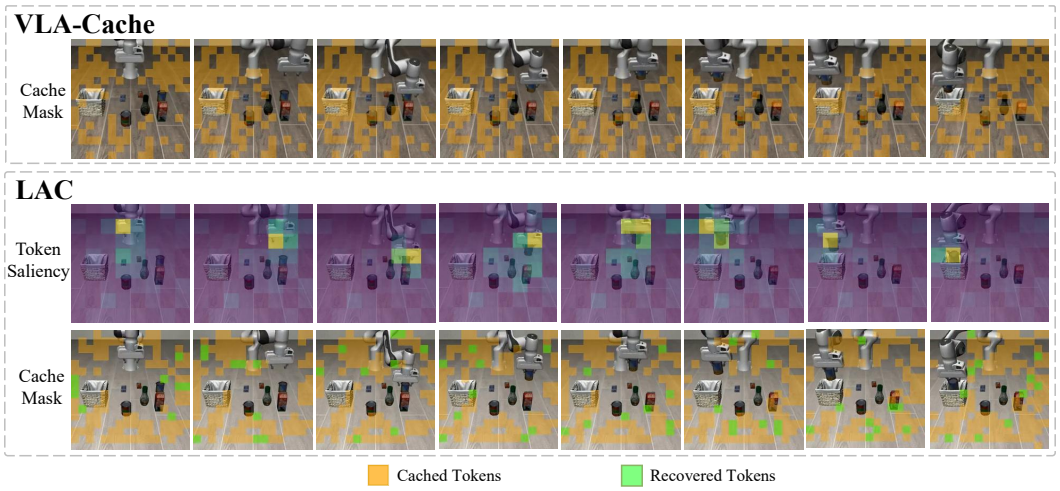


Figure 3: **Learned vs. rule-based caching on LIBERO.** Orange denotes cached tokens and green denotes recovered tokens. LAC recomputes task-relevant gripper regions, whereas the rule-based method incorrectly reuses the initially static target basket.

on LIBERO-Long, where LAC reaches 59.2%, exceeding SP-VLA by 5.0 points. With the stronger $\pi_{0.5}$ backbone, LAC retains a 96.4% average success rate while reducing normalized FLOPs to 88.1% and achieving a 1.22× speedup. These results demonstrate that LAC delivers the best overall accuracy–efficiency trade-off and generalizes across backbone architectures.

Qualitative Results on LIBERO. Figure 3 shows that LAC focuses recomputation on the moving gripper while caching the static background. In contrast, the rule-based method caches the initially static target basket; when the basket becomes relevant, the stale representation causes the robot to stall at its rim. This shows that task-optimized saliency is more reliable than fixed caching heuristics.

Effect of Token Reusing Ratio. Figure 4 compares methods under different token reusing or pruning rates. As the reuse ratio increases, LAC maintains stable performance, whereas pruning-based methods (e.g., SparseVLM) exhibit steep drops in success rate. Moreover, our CUDA inference time remains consistently lower, indicating genuine wall-clock efficiency rather than FLOPs reduction.

Ablation Studies. Table 2 evaluates LAC's components, two-stage training strategy, and optical-flow input. The Token Selector alone achieves an 82.2% success rate, demonstrating that learned token-level saliency captures important visuomotor cues. Adding the Reuse Predictor improves performance to 83.4% by adapting the cache ratio to scene dynamics. Stochastic recovery completes the full model and further raises success to 85.6%, improving robustness across task settings. Direct end-to-end training without Stage I initialization reduces success to 79.2%, confirming the impor-

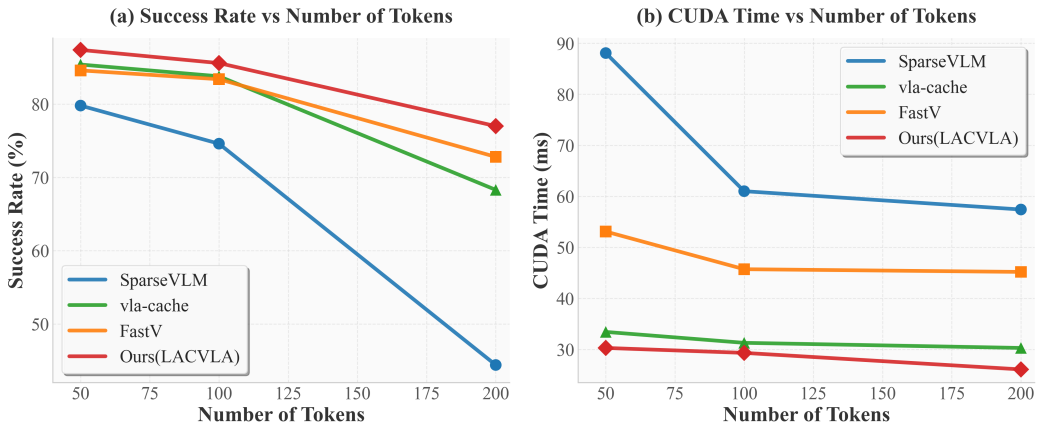


Figure 4: **Effect of token reusing/pruning ratio on LIBERO-Spatial.** Comparison of different methods under varying token reusing (or pruning) ratios. **Left:** Task success rate. **Right:** CUDA inference time. Our LAC maintains consistently high performance as the reuse ratio increases, while pruning-based methods such as SparseVLM degrade rapidly. In addition, LAC achieves the lowest CUDA latency across all settings, confirming its genuine wall-clock acceleration.

Table 2: Ablation of LAC components, training strategy, and input cue on LIBERO-Spatial.

Category	Method Variant	Success (%)	FLOPs (T) ↓	Time (ms) ↓
Component	Selector only	82.20	1.283	28.48
	+ Reuse Predictor	83.40	1.325	29.04
Training	w/o Stage I Init (Direct E2E)	79.20	1.377	29.32
Input Cue	RGB-only LAC (w/o Flow)	83.00	1.210	26.12
Full	**Full LAC**	**85.60**	**1.377**	**29.32**

tance of the attention-based saliency prior. The RGB-only variant reaches 83.0% with lower latency and FLOPs, while its accuracy gap shows the benefit of explicit motion cues. Together, these results demonstrate LAC's components, training strategy, and motion input.

Recovery Probability. On LIBERO-Spatial, varying p_{recover} from 0% to 20% improves success from 83.4% to 86.0% with only 0.64 ms additional latency. We use $p_{\text{recover}} = 10\%$ by default; full results and analysis are provided in Appendix D.3.

Table 3: **Results on the SIMPLER benchmark using CogAct as the base model.** We evaluate Visual Matching and Variant Aggregation across four manipulation tasks and report average success, FLOPs, and speedup.

SIMPLER	Method	PickCan	MoveNear	Drawer	DrawerApple	Average	FLOPs (%, ↓)	Speedup ↑
Visual Matching	Baseline (CogAct)	91.3%	85.0%	71.8%	50.9%	74.8%	100.0	1.00×
	FastV	92.6%	81.4%	69.8%	52.4%	74.1%	80.0	1.21×
	VLA-Cache	92.0%	83.3%	70.5%	51.6%	74.4%	81.0	1.37×
	MoLe-VLA	86.4%	80.2%	70.6%	50.4%	71.9%	47.4	1.53×
	Ours (LAC)	**92.3%**	**84.2%**	**72.7%**	**52.8%**	**75.5%**	**81.8**	**1.44×**
Variant Aggregation	Baseline (CogAct)	89.6%	80.8%	28.3%	46.6%	61.3%	100.0	1.00×
	FastV	91.4%	78.6%	27.6%	50.6%	62.1%	80.0	1.19×
	VLA-Cache	91.7%	79.3%	32.5%	45.8%	62.3%	82.6	1.37×
	MoLe-VLA	89.2%	79.5%	29.9%	46.2%	61.2%	46.3	1.49×
	Ours (LAC)	**92.1%**	**81.5%**	**31.2%**	**47.1%**	**63.0%**	**83.3**	**1.41×**

Main Results on SIMPLER. Table 3 demonstrates the strong control performance and efficiency of LAC across both SIMPLER evaluation protocols. In Visual Matching, LAC achieves the highest average success rate of 75.5%, improving upon the dense CogAct baseline while delivering a

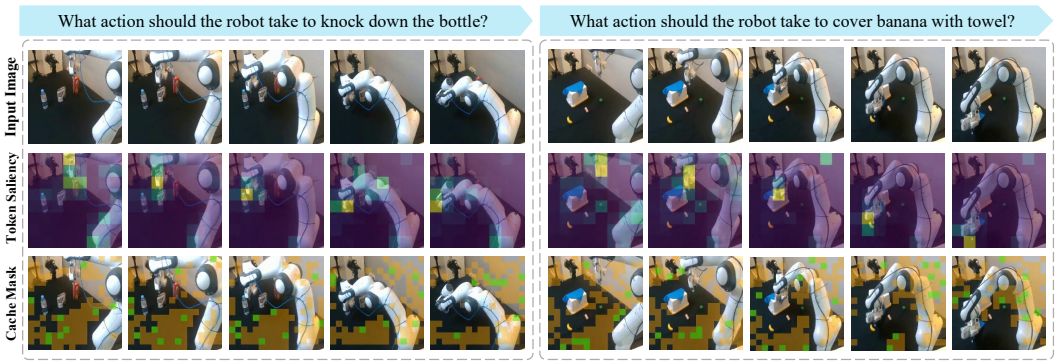


Figure 5: **Qualitative visualization of LAC on a real-world manipulation task.** The Token Saliency map identifies the moving end-effector and its interaction space as critical regions. The Cache Mask reuses static background tokens while recomputing task-relevant dynamics. Additional examples are provided in the Appendix.

1.44× wall-clock speedup. In the more challenging Variant Aggregation setting, LAC again obtains the highest average success rate of 63.0% and consistently improves performance across all four tasks, with a particularly notable gain on Drawer from 28.3% to 31.2%. Meanwhile, LAC reduces the FLOPs of CogAct to 81.8% and 83.3% in the two settings, respectively. These results demonstrate that LAC effectively removes redundant visual computation while preserving task-relevant information, enabling faster inference and stronger control performance across diverse evaluation conditions.

Table 4: Real-world robotic manipulation results on four tasks. Each rate uses 25 trials.

Method	Knock Crisp	Pick Mango	Cover Banana	Knock Bottle	Average	FLOPs (T) ↓	CUDA Time (ms) ↓
Baseline (OpenVLA)	48.0%	16.0%	24.0%	44.0%	33.0%	1.893	37.38
VLA-Cache	48.0%	12.0%	20.0%	40.0%	30.0%	1.534	33.36
Ours (LAC)	**52.0%**	**12.0%**	**24.0%**	**64.0%**	**38.0%**	**1.569**	**32.47**

4.2 Results on Real Robot

We validate our approach on a physical robot across four manipulation tasks, with quantitative results detailed in Table 4. LAC improves the average success rate by 5.0 points over the baseline while reducing FLOPs and inference latency.

Figure 5 provides qualitative insight into these results. The Token Saliency map confirms that LAC dynamically focuses computation on the robot's end-effector and its immediate interaction space. Accordingly, the Cache Mask reuses static background tokens while recomputing salient moving regions. This task-aware allocation not only explains the computational gains but also contributes to more robust performance in cluttered real-world environments.

5 Conclusion

This paper addresses the computational inefficiency of Vision-Language-Action (VLA) models for real-world robotics. We introduce LAC, a learnable, task-oriented framework with two lightweight, jointly optimized modules that dynamically determine which visual tokens to recompute or reuse from a cache. Experiments on simulation benchmarks and a physical robot show that LAC achieves up to a 1.76× inference speedup while improving task success rates over static caching and pruning methods. These results support our core hypothesis: adaptively reusing visual information can improve the efficiency, robustness, and overall performance of robotic control.

AI USE STATEMENT

Large Language Models (LLMs) were used to aid in the writing and polishing of the manuscript. Specifically, we used an LLM to assist in refining the language, improving readability, and ensuring clarity in various sections of the paper. The model helped with tasks such as sentence rephrasing, grammar checking, and enhancing the overall flow of the text.

It is important to note that the LLM was not involved in the ideation, research methodology, or experimental design. All research concepts, ideas, and analyses were developed and conducted by the authors. The contributions of the LLM were solely focused on improving the linguistic quality of the paper, with no involvement in the scientific content or data analysis.

The authors take full responsibility for the content of the manuscript, including any text generated or polished by the LLM. We have ensured that the LLM-generated text adheres to ethical guidelines and does not contribute to plagiarism or scientific misconduct.

ETHICS STATEMENT

This work does not involve human participants, animal subjects, personally identifiable information, or sensitive personal data. The simulation experiments use established robotics benchmarks, and the physical-robot experiments consist of object-manipulation tasks conducted under supervision in a controlled workspace. Consequently, institutional ethics approval was not required. We recognize that embodied AI systems may present physical-safety and dual-use risks when deployed outside controlled research settings. The proposed method is intended for research on computational efficiency and should not be interpreted as establishing the safety of autonomous deployment. The authors conducted this research in accordance with the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

We take several measures to facilitate the reproducibility of our results. All quantitative experiments are conducted using multiple random seeds and repeated evaluations to assess the robustness of the reported findings. We build on publicly available, open-source backbone models and checkpoints, and evaluate our method in the standardized and publicly accessible LIBERO and SIMPLER environments. The experimental protocols are described in Section 4, while implementation details, latency profiling, sensitivity analyses, and additional results are provided in the appendix. All hyperparameters and implementation details will be fully documented and released in a public GitHub repository.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. In *NeurIPS*, volume 35, pp. 23716–23736, 2022.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Chelsea Finn, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Jianjian Cao, Peng Ye, Shengze Li, Chong Yu, Yansong Tang, Jiwen Lu, and Tao Chen. MADTP: multimodal alignment-guided dynamic token pruning for accelerating vision-language transformer. In *CVPR*, pp. 15710–15719. IEEE, 2024.

Qingqing Cao, Bhargavi Paranjape, and Hannaneh Hajishirzi. Pumer: Pruning and merging tokens for efficient vision language models. In *ACL*, pp. 12890–12903, 2023.

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. In *ECCV*, volume 15139 of *Lecture Notes in Computer Science*, pp. 19–35. Springer, 2024.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. In *NeurIPS*, volume 36, pp. 49250–49267, 2023.

Danny Driess, Fei Xia, Apoorva Anand, Aakanksha Chowdhery, Wenlong Huang, et al. Palm-e: An embodied multimodal language model. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8469–8488. PMLR, 2023.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling. In *ACL*, pp. 320–335, 2022.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Anwen Hu, Haiyang Xu, Liang Zhang, Jiabo Ye, Ming Yan, Ji Zhang, Qin Jin, Fei Huang, and Jingren Zhou. mplug-docowl2: High-resolution compressing for ocr-free multi-page document understanding. In *ACL*, pp. 5817–5834, 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *ICLR*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. In *CoRL*, volume 270 of *Proceedings of Machine Learning Research*, pp. 4573–4602. PMLR, 2024.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. In *CoRL*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2024.

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

Junnan Li, Dongxu Li, Silvio Savarese, and Steven C. H. Hoi. BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19730–19742. PMLR, 2023.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024a.

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*, 2025a.

Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, et al. Vision-language foundation models as effective robot imitators. In *ICLR*. OpenReview.net, 2024b.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. In *CoRL*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3705–3728. PMLR, 2024c.

Yanwei Li, Chengyao Wang, and Jiaya Jia. Llama-vid: An image is worth 2 tokens in large language models. In *ECCV*, volume 15104 of *Lecture Notes in Computer Science*, pp. 323–340. Springer, 2024d.

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 *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=RwdGIIjPlC.

Yi Li, Yuquan Deng, Jesse Zhang, Joel Jang, Marius Memmel, Caelan Reed Garrett, Fabio Ramos, Dieter Fox, Anqi Li, Abhishek Gupta, and Ankit Goyal. HAMSTER: hierarchical action models for open-world robot manipulation. In *ICLR*. OpenReview.net, 2025b.

Haokun Lin, Haoli Bai, Zhili Liu, Lu Hou, Muyi Sun, Linqi Song, Ying Wei, and Zhenan Sun. Mope-clip: Structured pruning for efficient vision-language models with module-wise pruning error metric. In *CVPR*, pp. 27360–27370. IEEE, 2024.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. In *NeurIPS*, volume 36, pp. 44776–44791, 2023.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In *CVPR*, pp. 26286–26296. IEEE, 2024a.

Jiaming Liu, Mengzhen Liu, Zhenyu Wang, Pengju An, et al. Robomamba: Efficient vision-language-action model for robotic reasoning and manipulation. In *NeurIPS*, volume 37, pp. 40085–40110, 2024b.

Jiaming Liu, Hao Chen, Pengju An, Zhuoyang Liu, Renrui Zhang, Chenyang Gu, Xiaoqi Li, Ziyu Guo, Sixiang Chen, Mengzhen Liu, et al. Hybridvla: Collaborative diffusion and autoregression in a unified vision-language-action model. *arXiv preprint arXiv:2503.10631*, 2025a.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. RDT-1B: a diffusion foundation model for bimanual manipulation. In *ICLR*. OpenReview.net, 2025b.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and RT-X models : Open x-embodiment collaboration. In *ICRA*, pp. 6892–6903. IEEE, 2024.

Seongmin Park, Hyungmin Kim, Wonseok Jeon, Juyoung Yang, Byeongwook Jeon, Yoonseon Oh, and Jungwook Choi. Quantization-aware imitation-learning for resource-efficient robotic control. *arXiv preprint arXiv:2412.01034*, 2024.

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. *arXiv preprint arXiv:2501.09747*, 2025.

Yuzhang Shang, Mu Cai, Bingxin Xu, Yong Jae Lee, and Yan Yan. Llava-prumerge: Adaptive token reduction for efficient large multimodal models. In *ICCV*, pp. 22857–22867, 2025.

Leyang Shen, Gongwei Chen, Rui Shao, Weili Guan, and Liqiang Nie. Mome: Mixture of multimodal experts for generalist multimodal large language models. In *NeurIPS*, volume 37, pp. 42048–42070, 2024.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. In *RSS*, 2024.

Zachary Teed and Jia Deng. RAFT: recurrent all-pairs field transforms for optical flow. In *ECCV*, volume 12347 of *Lecture Notes in Computer Science*, pp. 402–419. Springer, 2020.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

Sicheng Wang, Sheng Liu, Weiheng Wang, Jianhua Shan, and Bin Fang. Robobert: An end-to-end multimodal robotic manipulation model. *arXiv preprint arXiv:2502.07837*, 2025.

Junjie Wen, Jinqiang Cui, Zhenjun Zhao, Ruixin Yan, Zhi Gao, Lihua Dou, and Ben M. Chen. Syreanet: A physically guided underwater image enhancement framework integrating synthetic and real images. In *ICRA*, pp. 5177–5183. IEEE, 2023.

Junjie Wen, Yichen Zhu, Jinming Li, Zhibin Tang, Chaomin Shen, and Feifei Feng. Dexvla: Vision-language model with plug-in diffusion expert for general robot control. *arXiv preprint arXiv:2502.05855*, 2025a.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE RA-L*, 2025b.

Junjie Wen, Yichen Zhu, Minjie Zhu, Zhibin Tang, Jinming Li, Zhongyi Zhou, Xiaoyu Liu, Chaomin Shen, Yaxin Peng, and Feifei Feng. Diffusionvla: Scaling robot foundation models via unified diffusion and autoregression. In *ICML*, 2025c.

Long Xing, Qidong Huang, Xiaoyi Dong, Jiajie Lu, Pan Zhang, Yuhang Zang, Yuhang Cao, Conghui He, Jiaqi Wang, Feng Wu, et al. Pyramiddrop: Accelerating your large vision-language models via pyramid visual redundancy reduction. *arXiv preprint arXiv:2410.17247*, 2024.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, et al. Vla-cache: Towards efficient vision-language-action model via adaptive token caching in robotic manipulation. *arXiv preprint arXiv:2502.02175*, 2025.

Senqiao Yang, Yukang Chen, Zhuotao Tian, Chengyao Wang, Jingyao Li, Bei Yu, and Jiaya Jia. VisionZip: Longer is better but not necessary in vision language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19792–19802, June 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Yang_VisionZip_Longer_is_Better_but_Not_Necessary_in_Vision_Language_CVPR_2025_paper.html.

Yang Yue, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang. Deer-vla: Dynamic inference of multimodal large language models for efficient robot execution. In *NeurIPS*, volume 37, pp. 56619–56643, 2024.

Jianke Zhang, Yanjiang Guo, Xiaoyu Chen, Yen-Jen Wang, Yucheng Hu, Chengming Shi, and Jianyu Chen. Hirt: Enhancing robotic control with hierarchical robot transformers. In *CoRL*, volume 270 of *Proceedings of Machine Learning Research*, pp. 933–946. PMLR, 2024a.

Rongyu Zhang, Menghang Dong, Yuan Zhang, Liang Heng, et al. Mole-vla: Dynamic layer-skipping vision language action model via mixture-of-layers for efficient robot manipulation. *arXiv preprint arXiv:2503.20384*, 2025.

Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, et al. Sparsevlm: Visual token sparsification for efficient vision-language model inference. *arXiv preprint arXiv:2410.04417*, 2024b.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models. In *CVPR*, pp. 1702–1713, 2025.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *RSS*, 2023.

Yifan Zhong, Xuchuan Huang, Ruochong Li, Ceyao Zhang, Zhang Chen, Tianrui Guan, Fanlian Zeng, Ka Num Lui, Yuyao Ye, Yitao Liang, et al. Dexgraspvla: A vision-language-action framework towards general dexterous grasping. *arXiv preprint arXiv:2502.20900*, 2025.

Zhongyi Zhou, Yichen Zhu, Minjie Zhu, Junjie Wen, Ning Liu, Zhiyuan Xu, Weibin Meng, Yaxin Peng, Chaomin Shen, Feifei Feng, et al. Chatvla: Unified multimodal understanding and robot control with vision-language-action model. In *EMNLP*, pp. 5377–5395, 2025.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. RT-2: vision-language-action models transfer web knowledge to robotic control. In *CoRL*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 2023.

APPENDIX

A LIMITATIONS

While LAC enables efficient adaptive inference, it presents two main limitations. First, our reliance on optical flow as a motion prior means that extreme visual conditions may affect the precision of the token selector. Second, in scenarios with rapid global visual changes or high-speed camera ego-motion, the policy will intelligently opt to recompute most tokens to preserve accuracy; consequently, the efficiency gains in such highly dynamic settings will naturally diminish compared to more stable environments.

B IMPLEMENTATION DETAILS

This section further details the technical implementation of our Learnable Adaptive Caching (LAC) framework within the Transformer's forward pass. At each inference step t, the binary mask $\mathbf{M}_t$ generated by our policy dictates the selective computation process. This is realized through the following modifications to the VLA decoder's forward pass:

- **Position Management and Attention Masking.** An internal state array tracks which tokens, as specified by $\mathbf{M}_t$, require recomputation. Tokens designated for caching preserve their positional encodings from the previous step. This is crucial as it allows the attention masks to be dynamically pruned, restricting the scope of the self-attention mechanism to the reduced set of active tokens and thus saving computation.
- **Conditional Rotary Embedding.** To introduce fresh positional information for the current timestep, rotary embeddings are applied exclusively to the recomputed tokens. Cached tokens bypass this operation, retaining prior states and avoiding redundant processing.
- **Dynamic Key-Value (KV) Cache Updates.** The KV cache at each Transformer layer l is updated dynamically. Newly computed tokens update their respective entries with $\{\mathbf{K}_t^l, \mathbf{V}_t^l\}$, while reused tokens inherit their values from the prior frame's cache, $\{\mathbf{K}_{t-1}^l, \mathbf{V}_{t-1}^l\}$. This partial update strategy, which mixes new and old KV entries, yields valid attention outputs. Transformer permutation invariance allows attention over any valid key-value set.

This entire mechanism is fully compatible with the standard KV caching used for autoregressive decoding. The most significant computational gain is realized while processing the multimodal prefix and generating the initial action token at each timestep. Subsequent action tokens still incur the standard autoregressive decoding cost, but they do not require rerunning the optical-flow estimator or the LAC decision modules.

C FORMAL EFFICIENCY ANALYSIS

We provide a theoretical complexity analysis to quantify the computational benefits of LAC. Let N denote the number of visual tokens per frame, D the embedding dimension, L the number of Transformer layers, and M the intermediate dimension of the Feed-Forward Network (FFN).

Policy Overhead. The overhead of our method stems from the fixed-cost optical flow estimation (RAFT) and the lightweight CNN selector. This cost scales linearly with the input image resolution (H, W) and is independent of the Transformer depth or sequence length:

$$\mathcal{C}_{\text{policy}} \approx \mathcal{O}(H \cdot W \cdot C_{\text{cnn}}). \tag{12}$$

Since this overhead is computed only once per frame and does not scale with the model depth L, it remains negligible compared to the backbone computation.

Baseline Computational Cost. In a standard VLA Transformer layer, the computational cost consists of Multi-Head Self-Attention (MSA) and Feed-Forward Networks (FFN). For a sequence of N tokens, the FLOPs per layer are:

$$\mathcal{C}_{\text{base}} \approx \underbrace{4ND^2 + 2N^2D}_{\text{MSA}} + \underbrace{2NDM}_{\text{FFN}}. \tag{13}$$

Here, $4ND^2$ accounts for QKV and output projections, $2N^2D$ represents the attention score computation and aggregation, and $2NDM$ accounts for the two linear transformations in the FFN (projection $D \to M$ and $M \to D$).

LAC Computational Cost. Our method processes only a subset of active tokens $N_{\text{act}} = (1-\rho)N$, where ρ is the cache ratio.

- **Projections & FFN:** We compute linear projections and FFNs strictly for active tokens.
- **Attention:** Active queries (N_{act}) attend to all active and cached tokens (N total).

The reduced cost per layer is:

$$\mathcal{C}_{\text{lac}} \approx \underbrace{4N_{\text{act}}D^2 + 2(N_{\text{act}} \cdot N)D}_{\text{Partial MSA}} + \underbrace{2N_{\text{act}}DM}_{\text{Partial FFN}}. \tag{14}$$

Theoretical FLOPs Reduction. The reduction in FLOPs per layer is derived by subtracting the LAC cost from the baseline:

$$\Delta\text{FLOPs}_{\text{layer}} = \mathcal{C}_{\text{base}} - \mathcal{C}_{\text{lac}}. \tag{15}$$

Integrating the overhead, the total efficiency gain for the model is:

$$\Delta\text{FLOPs}_{\text{total}} \approx \sum_{l=1}^{L} \Delta\text{FLOPs}_{\text{layer}} - \mathcal{C}_{\text{policy}}. \tag{16}$$

This analysis demonstrates that LAC significantly reduces the theoretical computational complexity, at the cost of a minimal, constant policy overhead.

C.1 End-to-End Latency Profiling

We additionally profile the complete inference path to verify that the reported speedup includes the overhead of the learned policy. As shown in Table 5, optical-flow estimation and the selector/predictor together require 4.4 ms, while selective VLA computation reduces the backbone forward pass to 25.1 ms. The resulting end-to-end latency is 29.5 ms, corresponding to a 1.76× speedup over the 51.9 ms OpenVLA baseline.

Table 5: End-to-end latency breakdown on LIBERO, including all LAC overheads.

Component	**OpenVLA (ms)**	**LAC (ms)**
RAFT-small	–	3.2
Selector and reuse predictor	–	1.2
VLA forward	51.9	25.1
End-to-end total	**51.9**	**29.5**
Speedup	**1.00×**	**1.76×**

D Additional Experiment Results

D.1 Real-World Robot Tasks

Figure 6 visualizes our policy's behavior on two real-world tasks: *"pick up the apple and place it in the box"* (left) and *"knock down the crisp"* (right). The learned Token Saliency (middle row) demonstrates the model's ability to effectively distinguish between important and unimportant visual regions. High saliency scores are concentrated on the robot's end-effector and the interaction targets, while the static background is assigned low importance. Consequently, the Cache Mask (bottom row) reveals an efficient strategy: unimportant visual regions are cached (marked in orange), while the critical, high-saliency regions remain unmasked for active recomputation. Additionally, green tiles appear within the cached regions, illustrating the stochastic recovery mechanism that randomly updates a fraction of tokens to ensure robustness.

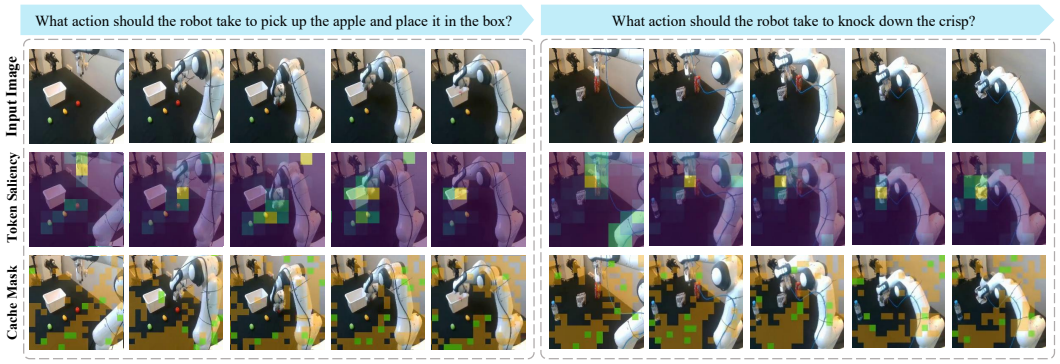


Figure 6: **Qualitative visualizations of the learned policy (LAC) on real-world tasks.** The examples show the policy correctly identifying task-relevant regions for recomputation. (Top) Input image sequence. (Middle) Learned Token Saliency map. (Bottom) Resulting Cache Mask, with orange for cached tokens and green for stochastically recovered tokens.

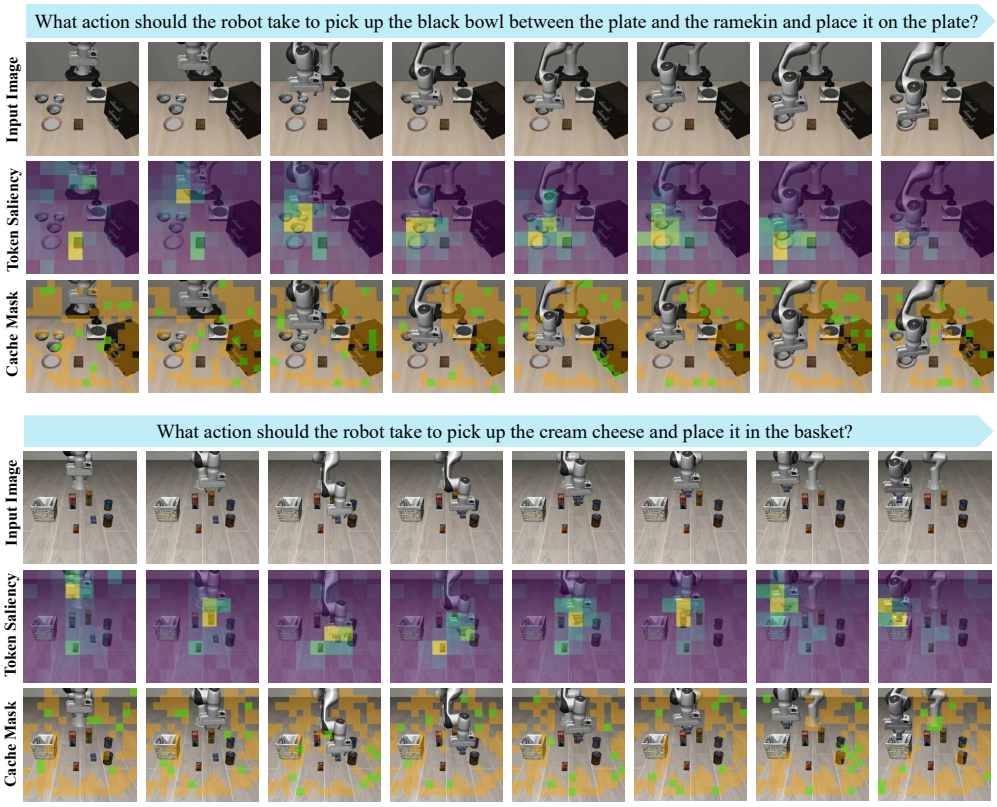


Figure 7: **Qualitative results on the LIBERO benchmark (part 1).** We visualize the behavior of LAC on the first set of manipulation tasks. (Top) Input image sequence; (Middle) Learned Token Saliency map; (Bottom) Generated Cache Mask (orange for cached, green for stochastically recovered, and unmasked regions for active). High saliency is consistently assigned to task-relevant regions like the gripper and target object, which are actively recomputed.

D.2 LIBERO Benchmark

Figures 7 and 8 present additional qualitative visualizations across four diverse tasks from the LIBERO benchmark. Consistent with our real-world observations, the learned policy accurately

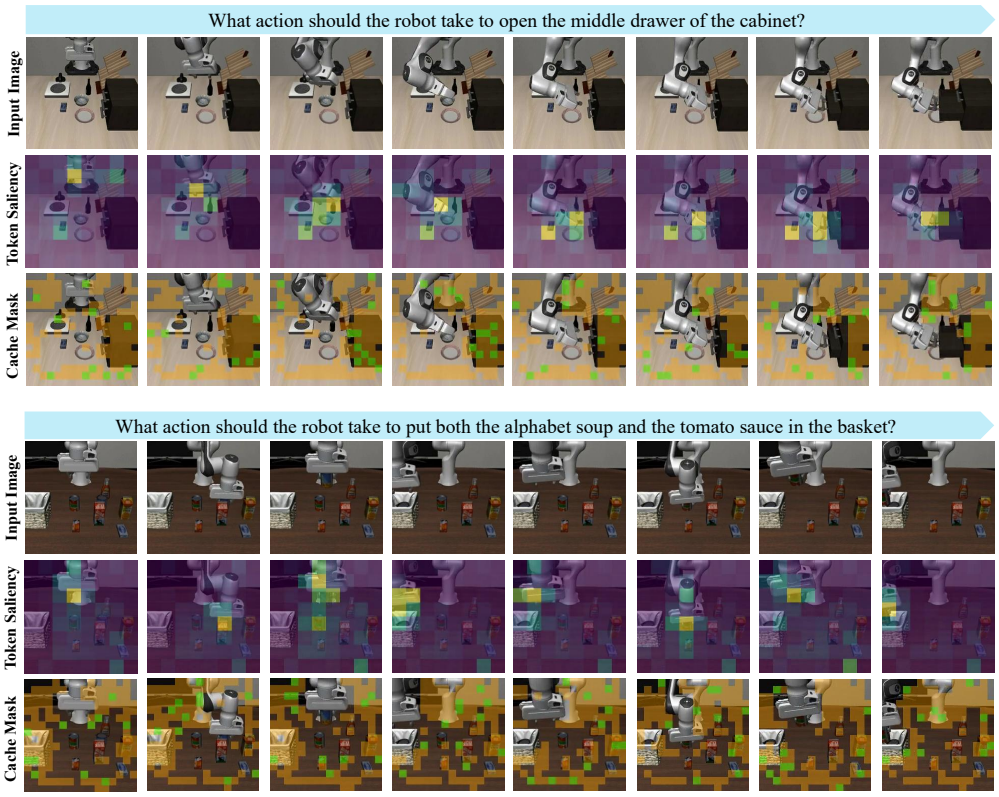


Figure 8: **Qualitative results on the LIBERO benchmark (part 2).** Visualization on additional tasks. Similar to Part 1, the policy successfully caches the unimportant visual regions (orange) while actively re-computing the moving agents and interaction targets (shown as unmasked). Green tiles indicate tokens updated via the stochastic recovery mechanism, demonstrating the robustness of our adaptive caching strategy.

identifies task-relevant dynamics: high saliency scores are assigned to the robot gripper and target objects, while the static environment is cached for reuse. The resulting Cache Masks in both figures confirm that LAC efficiently allocates computation by re-computing critical regions (unmasked) and retrieving the static background from the cache (orange), with occasional updates provided by stochastic recovery (green) to prevent error accumulation.

D.3 Sensitivity Analysis on Stochastic Recovery Probability

We analyze the sensitivity of LAC to the stochastic recovery probability p_{recover} on LIBERO-Spatial. All settings use the same evaluation protocol; only the recovery probability is varied. Table 6 reports task success, end-to-end CUDA latency, and computational cost.

Table 6: Sensitivity to the stochastic recovery probability on LIBERO-Spatial.

p_{recover}	**Success Rate (%)**	**CUDA Time (ms)**	**FLOPs (T)**
0% (w/o Recovery)	83.40	29.04	1.325
5%	84.20	29.13	1.350
10% (Default)	**85.60**	**29.32**	**1.377**
20%	86.00	29.68	1.430

Robustness. Performance improves smoothly with p_{recover} and remains above 84% for every nonzero setting, showing no abrupt degradation within the tested range. **Low Recovery Overhead.** Increasing p_{recover} from 0% to 20% adds only 0.64 ms of end-to-end latency (29.04 ms to 29.68 ms),

since recovery requires no additional network forward pass. **Default Operating Point.** We use $p_{\text{recover}} = 10\%$ because it captures most of the gain: relative to disabling recovery, success improves by 2.2 points for 0.28 ms additional latency, whereas 20% adds another 0.4 point at higher computational cost.

E Extended Discussion and In-depth Analysis

E.1 Why Can Selective Caching Outperform Dense Inference?

A natural question is why LAC, despite approximating dense inference through historical-state reuse, can obtain a higher task success rate than the full-computation baseline (e.g., $+1.9$ points on LIBERO with OpenVLA and $+5.0$ points in our real-robot evaluation). Importantly, dense computation is not an upper bound on closed-loop control performance: recomputing every visual state preserves all available information, but it also propagates every task-irrelevant frame-to-frame fluctuation. Our results suggest that the gain arises from *which representations are stabilized and which are refreshed*, rather than from performing less computation by itself.

- **Implicit spatio-temporal regularization.** Static regions can exhibit small changes caused by sensor noise, illumination variation, reflections, or camera vibration. Dense inference re-encodes these nuisance variations at every control step. Reusing their historical states instead stabilizes the corresponding representations across adjacent observations, acting as an implicit temporal low-pass filter. This can reduce sensitivity to transient background distractions while preserving newly computed features in regions whose state is changing.
- **Task-aware emphasis on interaction dynamics.** LAC concentrates recomputation on the active workspace, including the end-effector, manipulated object, and nearby interaction region. Cached tokens are not removed: their historical keys and values remain available to attention. The resulting allocation therefore preserves global scene context while dedicating fresh computation to control-relevant changes. The qualitative examples in Figures 3, 6, 7, and 8 are consistent with this interpretation.
- **Alignment with the downstream control objective.** Attention magnitude, visual contrast, or other hand-designed saliency scores are only proxies for control relevance. In contrast, LAC trains its selection policy using the downstream VLA loss, so a token is favored for recomputation when refreshing it helps action prediction. This distinction explains why a learned selector can preserve task-critical cues more reliably than a rule-based policy under the same or a more aggressive reuse budget.
- **Protection against long-horizon staleness.** Stochastic recovery refreshes a small fraction of otherwise cached tokens, allowing the system to discover changes that were not selected initially and preventing errors from accumulating indefinitely. Table 6 supports this role: increasing p_{recover} from 0% to 10% improves success by 2.2 points at only 0.28 ms additional latency. This mechanism is best understood as stochastic state refresh rather than conventional feature dropout, because recovered tokens are recomputed rather than removed.

The component ablation in Table 2 provides additional evidence for this explanation: adapting the reuse budget improves over the selector-only variant, and stochastic recovery further improves the result. Nevertheless, these observations should not be interpreted as a universal guarantee that approximation improves accuracy. For example, LAC retains but does not exceed the dense $\pi_{0.5}$ result in Table 1. We therefore view the gains on OpenVLA, CogAct, and the physical robot as evidence that task-aware temporal stabilization can improve robustness in some closed-loop regimes, not as a claim that reduced computation is intrinsically more accurate.

E.2 Methodological Differences from Representative VLM Token Reduction

Although VLM acceleration methods such as FastV (Chen et al., 2024), SparseVLM (Zhang et al., 2024b), and VisionZip (Yang et al., 2025) also reduce visual-token computation, LAC addresses a different redundancy structure. Table 7 summarizes the main differences.

Table 7: Methodological comparison between representative VLM token-reduction methods and LAC. "Unavailable" means that a discarded token is no longer explicitly accessible to subsequent layers in the same forward pass; it does not imply removal from the original image.

Dimension	FastV	SparseVLM	VisionZip	LAC
Primary setting	Single-image/video VLM inference	Single-image/video VLM inference	Single-/multi-turn VLM inference	Closed-loop robot control
Redundancy used	Layer-wise visual-token redundancy	Text-guided visual-token redundancy	Intra-input visual-token redundancy	Cross-frame temporal redundancy
Token operation	Prune after early-layer attention scoring	Prune and recycle selected tokens	Select dominant tokens and merge the remainder	Reuse historical KV states or recompute them
Later access	Pruned tokens are unavailable to deeper layers	Removed information is compressed into recycled tokens	Non-dominant tokens are compressed into contextual tokens	Every token position retains a state and can be refreshed later
Selection signal	Early-layer attention	Relevant text tokens and self-attention	Vision-encoder attention and token similarity	Optical flow, learned saliency, and downstream VLA loss
Reuse-budget policy	Preset layer and pruning ratio	Adaptive layer-wise sparsification	Preset dominant/contextual token budgets	Per-frame learned cache-ratio prediction
Temporal state across control steps	Not explicitly modeled	Not explicitly modeled	Not explicitly modeled	Explicit layer-wise cross-frame cache

The central distinction is therefore *non-destructive temporal reuse* rather than a new single-frame pruning score. A VLM pruning method may remove an apparently unimportant object from the deeper layers of the current forward pass. This is often acceptable for answering a question about a fixed image, but it can be risky in manipulation: an idle receptacle at timestep t may become the destination at timestep $t + 10$. LAC retains the object's historical representation and can recompute it when motion or task-conditioned saliency changes.

This difference also affects wall-clock behavior. Layer-wise pruning or merging can require repeated ranking, indexing, clustering, or tensor reshaping. For the relatively short action-generation workloads considered here, these operations need not translate FLOPs savings into latency savings: Table 1 shows that our SparseVLM implementation reduces normalized FLOPs to 75.5% but runs at $0.62\times$ the dense-baseline speed. LAC instead makes one frame-level reuse-budget decision and performs partial KV updates throughout the backbone. After including optical flow and all policy overheads, it achieves a measured $1.76\times$ end-to-end speedup. This comparison is specific to our implementation and hardware setting and should not be read as a general claim that pruning kernels must be slower.

E.3 Why Perception-Side Prefilling Remains Important

Autoregressive decoding is a major bottleneck in long-form language generation and can also be important for VLA policies. Methods such as FAST (Pertsch et al., 2025) improve the output side by compressing action sequences into more efficient token representations. LAC does not challenge this motivation. Instead, it targets a complementary, repeatedly incurred cost on the input side.

- **VLA control has a different token-length regime from long-form generation.** At each OpenVLA control step, the multimodal prefix contains hundreds of visual tokens plus the language instruction, whereas the output contains only about seven discrete action tokens. Thus,

$$N_{\text{visual}} + N_{\text{text}} \gg N_{\text{action}} \approx 7. \tag{17}$$

The first action-token computation must process the long multimodal context, while each subsequent decoding step appends only a short token to the existing KV cache. Consequently, reducing repeated visual processing can materially reduce the latency of each control query even when decoding remains non-negligible.

- **Closed-loop interaction repeatedly pays the perception cost.** A language prompt is typically prefetched once before a long response, but a robot receives a new observation throughout task execution. Each observation must be incorporated before the policy can react to object motion, contact, or external disturbance. Action chunking can reduce the frequency of policy calls, but whenever the policy consumes a fresh frame, the visual context must still be updated. LAC exploits the fact that most of this new frame is often redundant with the previous one.
- **The target is shared across heterogeneous action decoders.** "Prefill" and "decode" precisely describe autoregressive OpenVLA, whereas CogAct and $\pi_{0.5}$ use diffusion- and flow-based action generation, respectively. All three architectures nevertheless condition action generation on visual representations. Cross-frame visual-state reuse targets this shared conditioning cost without requiring a particular action decoder.
- **Input- and output-side accelerators are complementary.** FAST changes how an action trajectory is represented and generated; LAC changes how a new observation is incorporated. Reducing either term lowers end-to-end latency, and optimizing both avoids leaving the other term as the limiting component. In principle, an efficient action tokenizer or action-chunking method can therefore be combined with LAC, because neither requires the other to preserve redundant visual computation or to use an inefficient output representation.

Our current latency measurements support the practical value of the perception-side intervention without claiming that prefilling universally dominates every VLA architecture. As reported in Table 5, selective visual computation reduces the measured VLA forward time from 51.9 ms to 25.1 ms; after adding 4.4 ms for optical flow and the learned policy, the end-to-end latency remains 29.5 ms. A hardware-specific separation of visual encoding, first-token multimodal processing, and subsequent action generation would provide an even finer attribution, but the end-to-end measurement already demonstrates that the targeted computation is sufficiently large to yield a real $1.76\times$ wall-clock improvement.

E.4 Robustness to Imperfect Optical Flow

Although LAC uses RAFT-small to expose motion explicitly, its caching policy is not a flow-only gating rule. Two design properties make optical flow an auxiliary motion prior rather than a single point of failure.

- **Joint appearance-and-motion input.** Both the Cached Token Selector and Cache Ratio Predictor operate on
$$V_t = [I_t; O_t], \tag{18}$$
where the RGB observation I_t and optical flow O_t are concatenated channel-wise. The decision modules therefore learn joint appearance-and-motion features. When flow is weak in a textureless region or locally noisy under illumination variation or motion blur, the RGB pathway still provides object appearance, spatial layout, and interaction-context cues. Conversely, flow supplies a direct and inexpensive indication of change when RGB appearance alone is ambiguous. This complementary input design is substantially more robust than thresholding flow magnitude to determine which tokens to cache.
- **Strong flow-free operating point.** The RGB-only ablation in Table 2 removes optical flow entirely, representing a deliberately conservative test of whether useful caching decisions can be learned from appearance alone. It still obtains an 83.0% success rate on LIBERO-Spatial, only 2.6 points below full LAC, while further reducing computation from 1.377 T to 1.210 T FLOPs and latency from 29.32 ms to 26.12 ms. Thus, LAC retains a strong and efficient control policy even without an explicit motion prior; RAFT improves the operating point but is not required for the selector architecture to function.

- **Adaptive compute allocation under large apparent motion.** The Cache Ratio Predictor does not enforce a fixed reuse rate. It selects a cache ratio from a discrete set for every observation and is trained jointly with the downstream task loss. Inputs exhibiting large or widespread apparent motion—including global motion responses that may arise under difficult visual conditions—favor lower reuse budgets, causing more visual tokens to be freshly recomputed. The policy can therefore exchange part of its efficiency gain for updated visual evidence instead of being forced to reuse a fixed fraction of potentially stale states.

Together, these properties explain why LAC should not be characterized as critically dependent on a perfectly accurate optical-flow field. RAFT-small provides a useful motion prior that improves task performance, whereas RGB semantics, task-loss supervision, adaptive ratio prediction, and stochastic recovery provide independent safeguards for stable caching decisions.