Reducing Temporal Redundancy for Efficient Vision-Language-Action Inference

Author Names Omitted for Anonymous Review

***Abstract*—Vision-Language-Action (VLA) models exhibit strong generalization for robotic manipulation, yet their high inference latency limits real time deployment. We identify two primary sources of temporal redundancy in existing VLA pipelines: repeated visual encoding of highly similar consecutive frames and multi step iterative sampling in diffusion based policies. To address this, we propose a system level acceleration strategy that reduces computation in both perception and action generation. On the perception side, we incrementally update only tokens corresponding to dynamic scene regions instead of re-encoding entire frames. On the policy side, we compress diffusion sampling into a compact 2-step schedule through efficiency oriented training while preserving action precision. Experiments on Libero, RobotWin, and Real Robot Platforms demonstrate over $2\times$ speedup while maintaining high performance, achieving up to 98% success rate on general manipulation benchmarks. Our codes will be released on Github.**

***Index Terms*—Vision-Language-Action Models, Robotic Manipulation, Diffusion Policy, Efficient Inference, Real-Time Robotics**

I. Introduction

The rapid advancement of Vision-Language-Models (VLMs) has driven the emergence of general purpose Vision-Language-Action (VLA) agents capable of interpreting natural language instructions and executing multi step physical tasks [23, 1, 17, 27]. Systems such as RT-2, OpenVLA, and π_0, built on Internet scale multimodal pre training, have significantly advanced robotic autonomy [43, 16, 3]. However, a substantial gap still remains between their impressive capabilities and real world deployment. Modern VLA pipelines rely heavily on large Vision Transformer (ViT) backbones and iterative inference mechanisms such as diffusion or flow matching [12, 9, 19], introducing considerable latency. This latency directly conflicts with the high frequency, closed loop control required for safe and responsive manipulation in dynamic environments [41, 11]. As a result, efficient inference is no longer merely desirable but absolutely essential for practical robotic deployment [43].

The need for low latency inference has motivated a growing body of research on VLA acceleration. However, most existing approaches focus on optimizing isolated components, resulting in limited system level gains. This fragmented strategy often shifts the computational bottleneck rather than eliminating it. For instance, methods such as TinyVLA [32] and DeeR-VLA [37] design lightweight architectures but are difficult to apply to large pre-trained models. Meanwhile, approaches like VLA-Cache [33] exploit visual redundancy through token caching, yet primarily target the perception module. The memory overhead of the LLM core and the iterative computation of the action head remain largely unaddressed. Furthermore, many existing acceleration studies are evaluated mainly in simulated environments and concentrate on improving VLM components, leaving the efficiency of action generation largely unexplored. Consequently, current approaches fall short of addressing the end to end latency of modern VLA agents at the system level[31, 30, 26]. To understand the source of inefficiency in VLA inference, we analyze the temporal behavior of both perception and policy modules. As shown in Fig. 1, the perception side Fig. 1(a,b) reveals a strong temporal consistency in visual representations. Visual tokens extracted from adjacent frames exhibit extremely high cosine similarity across multiple ViT layers, with most values consistently above 0.98. In addition, token dissimilarity maps show clear spatial sparsity: only a small fraction of tokens change noticeably between steps, while the majority remain nearly unchanged over time. These results indicate substantial temporal redundancy in streaming robotic perception. On the policy side Fig. 1(c), we examine the singular value spectrum of velocity trajectories in flow matching. The spectrum decays rapidly, with energy concentrated in the top two dominant components. This suggests that velocity evolution follows a low dimensional structure. As a result, multiple diffusion iterations mainly produce highly correlated refinements rather than independent directional updates.

Taken together, these findings expose a systemic structural redundancy across the entire VLA pipeline, motivating a unified acceleration strategy rather than isolated module level optimizations. Building directly upon the above observations, we propose a unified system level acceleration framework for flow based VLA models that reduces temporal redundancy in both perception and action generation. On the perception side, instead of re-encoding entire frames at every step, we incrementally update only tokens corresponding to dynamic regions between adjacent frames, effectively preserving static representations through token reuse; empirically, most scenes require recomputing only about 60% of tokens per frame. On the policy side, we exploit the low rank, linearly correlated structure of flow velocities and reconstruct the diffusion inference process into a compact two step schedule through efficiency oriented training, significantly reducing iterative computation while maintaining trajectory precision in practice.

Extensive experiments on LIBERO, RobotWin, and Real World Robotic Platforms demonstrate that our approach achieves more than $2\times$ end to end inference speedup with negligible performance degradation. In particular, our method

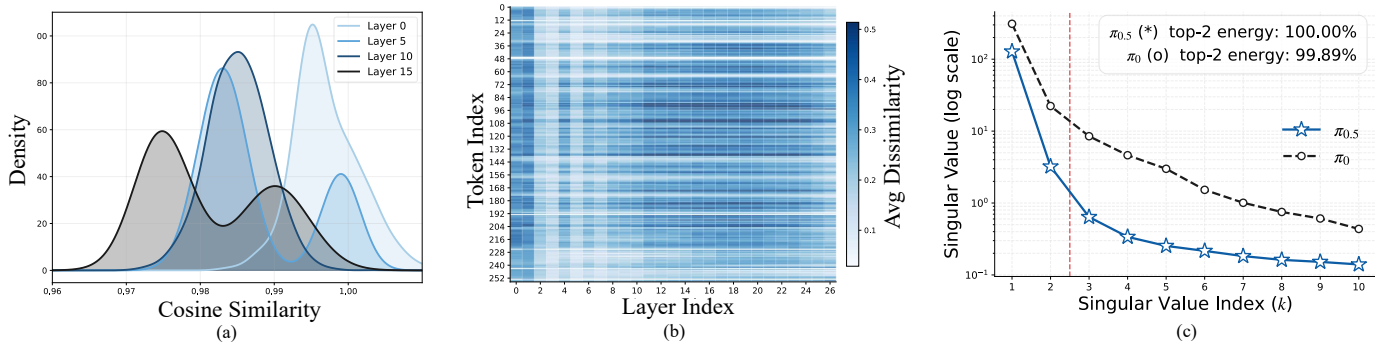

Fig. 1. **Structural redundancy analysis in perception and policy dynamics.** (a) Distribution of cosine similarity between visual tokens of consecutive frames across ViT layers. Most tokens exhibit near unity similarity (> 0.98), revealing strong temporal redundancy in streaming robotic perception. (b) Evolution of token dissimilarity across layers. Only a sparse subset of spatial tokens show significant deviation, supporting selective token refresh. (c) Intrinsic dimension analysis of flow matching velocity fields. Singular value spectra indicate that most energy is concentrated in the top two components, suggesting that velocity evolution lies in a low dimensional subspace. The rapid decay of lower index singular values implies strong linear correlation among velocity updates across time steps, allowing multiple diffusion steps to be approximated with a small number of updates. Together, these observations motivate our approach to exploit temporal redundancy in both perception and policy, enabling efficient VLA inference through selective token reuse and step-compressed policy execution.

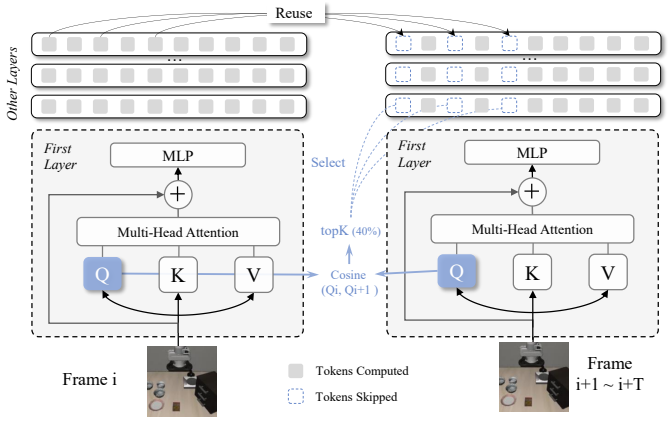

Fig. 2. **Temporal token reuse with cosine similarity selection.** Only a subset of tokens is updated based on cosine similarity, while others reuse cached representations. Index selection is performed in the first layer and reused across subsequent layers.

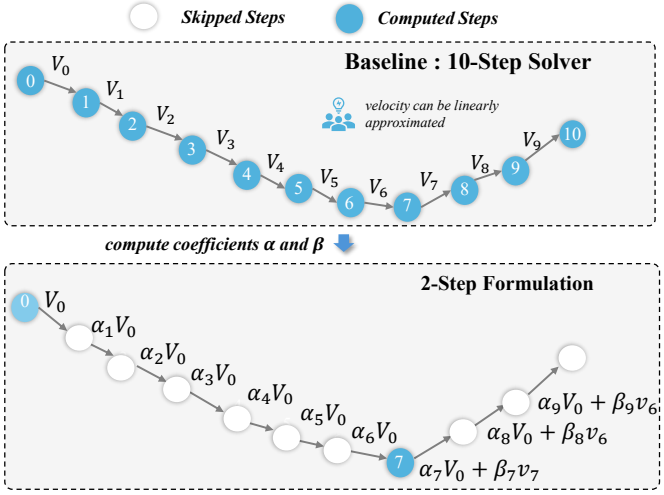

Fig. 3. **Flow step compression via low rank velocity structure.**

maintains comparable task accuracy while achieving nearly $2.6\times$ end to end speedup in simulation and $1.6\times$ acceleration on Real Robot Platforms. These results demonstrate that real time deployment of large scale VLA models is attainable without compromising manipulation performance. Our main contributions are summarized as follows:

- We identify previously overlooked structural redundancies in VLA inference, including strong temporal consistency in visual tokens across adjacent frames and low rank velocity dynamics in flow based action generation.
- Based on these insights, we propose a unified system level acceleration framework for flow based VLA models that reduces perception redundancy across adjacent frames through selective token updates, and reduces policy computation through a compact two-step flow inference scheme.
- We validate the proposed approach across simulation benchmarks (LIBERO, RobotWin) and real word robotic platforms, achieving over $2\times$ end to end inference speedup while maintaining comparable task performance.

II. Related Work

A. Vision-Language-Action Models for Manipulation

Vision-Language-Action (VLA) models aim to unify perception, language understanding, and control within a single framework, thereby enabling robots to execute diverse tasks specified by natural language instructions [29]. Recent works increasingly leverage large scale vision language pre training to endow robots with strong semantic grounding and generalization. Representative systems such as RT-2 [43], OpenVLA [16], and π_0 [4] adopt transformer based architectures and Internet scale data to directly map multimodal observations to actions, demonstrating consistently impressive zero shot and multi task manipulation capabilities.

Despite their success, most existing VLA systems are primarily designed with a focus on task generality and semantic alignment rather than real time efficiency. In practice, these models often rely on heavy vision backbones (e.g., SigLIP or ViT Huge) [38] and complex policy heads, resulting in high inference latency that hinders effective closed loop control [32]. Our work complements this line of research by directly targeting efficient inference for VLA models without sacrificing their general purpose nature.

B. Efficient Visual Perception for Robotics

Efficient perception has long been a central topic in robotics, with early works focusing on lightweight convolutional architectures and feature reuse[13, 39]. More recently, Vision Transformers (ViTs) have become dominant due to their strong representation capacity, but their quadratic complexity with respect to token count makes them computationally expensive for high frequency control[10, 21].

Several approaches have explored reducing the cost of ViT inference via token pruning, token merging, or early exiting. While effective in static or single image settings, these methods typically process each frame independently and fail to fully exploit the strong temporal continuity present in robotic video streams. In contrast, our approach explicitly leverages temporal redundancy by reusing visual tokens across consecutive frames and incrementally updating only regions directly affected by scene changes, enabling efficient streaming perception for real time manipulation.

C. Temporal Modeling and Incremental Inference

Temporal modeling is commonly addressed by recurrent networks, temporal convolutions, or video transformers that jointly process multiple frames[36, 2]. However, these methods often significantly increase computational cost and memory usage, making them generally less suitable for high frequency online control in practice.

Incremental inference has been explored in areas as video understanding and streaming transformers, where cached representations are reused across time steps. These techniques demonstrate that significant redundancy exists in consecutive inputs. Our method builds on this insight but is tailored to the robotics setting: rather than caching entire frames or sequences, we selectively reuse spatial tokens while maintaining feature consistency through targeted training, ensuring stable control under partial feature updates.

D. Diffusion and Flow Based Policies for Control

Diffusion and flow matching models have recently emerged as powerful policy representations for continuous control and robotic manipulation, due to their ability to model complex, inherently multi modal action distributions[15, 19, 8]. Prior works consistently show that iterative denoising processes can produce high quality trajectories, but often at the cost of multiple forward passes per control step, inevitably leading to high latency in practice overall.

To address this limitation, several studies investigate accelerated sampling strategies, such as reducing the number of diffusion steps or distilling iterative processes into fewer steps[25]. Our work follows this direction but focuses on preserving control precision under aggressive step compression. By combining a compact sampling schedule with an efficiency oriented training recipe, we achieve fast action generation suitable for real time closed loop control[22, 28].

III. Methodology

We propose an efficient acceleration method for Vision-Language-Action (VLA) systems that reduces redundant computation in both perception and action generation. Our method introduces temporally aware visual token reuse and a compressed 2-step flow matching policy, enabling significant latency reduction without altering the backbone architecture. The resulting framework supports real time closed loop control while preserving task performance.

A. Preliminaries: Vision-Language-Action Models

Vision-Language-Action (VLA) models aim to directly map multimodal observations to continuous control actions. Given an observation $o = (I, L, s)$ consisting of image I, language tokens L, and robot state s, a VLA policy predicts an action sequence $a \in \mathbb{R}^{H \times D}$ over a horizon H. Modern VLA systems typically adopt a transformer based architecture composed of: a vision encoder (e.g., SigLIP/ViT) that converts images into spatial tokens, a large language backbone that fuses visual and textual embeddings a policy head that generates actions via diffusion or flow matching inference.

In flow matching based policies, action generation is formulated as solving an ordinary differential equation (ODE):

$$\frac{dx_t}{dt} = v_\theta(x_t, t, o), \tag{1}$$

where x_t denotes the noisy action at time $t \in [0, 1]$, and v_θ is a learned velocity field conditioned on observation. Inference typically starts from Gaussian noise $x_1 \sim \mathcal{N}(0, I)$ and integrates backward to $t = 0$ using a numerical solver such as Euler discretization:

$$x_{t+\Delta t} = x_t + \Delta t \, v_\theta(x_t, t, o). \tag{2}$$

In practice, $N = 8 \sim 10$ solver steps are used per control cycle to ensure action precision. Each step requires a forward pass through the language backbone and policy head, making the total inference cost scale linearly with N. Under high frequency closed loop control, this iterative structure becomes a major latency bottleneck.

B. Temporal Token Reuse

As illustrated in Fig. 2, we adopt a simple odd even chunking strategy to identify tokens requiring updates. On odd chunks, we perform a full forward pass and cache key projections as reference representations. On even chunks, we compute cosine similarity between current and previously cached keys:

$$\mathrm{sim}(i) = \cos(K_t^{(i)}, K_{\mathrm{ref}}^{(i)}), \tag{3}$$

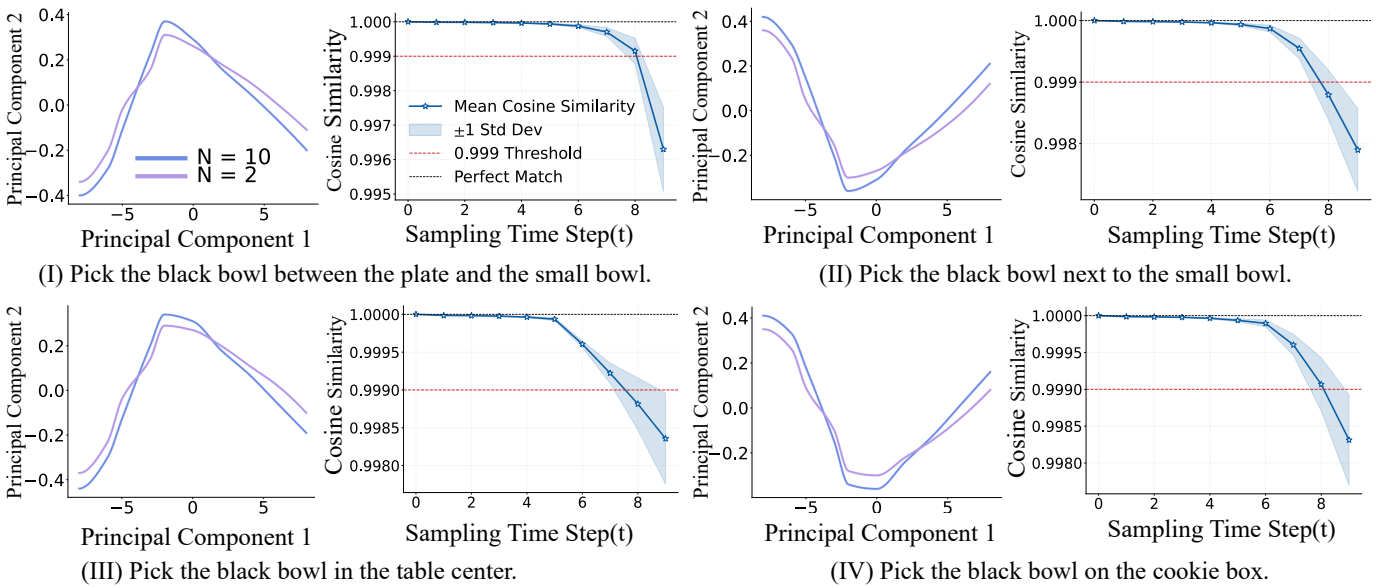

Fig. 4. **Flow matching step compression analysis.** For four representative manipulation tasks from the Libero Spatial benchmark, we visualize the PCA projected action trajectories together with the cosine similarity between intermediate velocity fields and the initial velocity v_0. The compressed 2-step solver produces trajectories that remain geometrically consistent with those generated by the original 10-step solver in the latent space. Meanwhile, the cosine similarity stays close to one across most sampling steps, revealing strong directional consistency in the velocity dynamics and explaining why aggressive step compression is possible.

and select the top-ρN tokens with lowest similarity for update, where ρ controls the update ratio.

As shown in Fig. 1(a,b), token similarity patterns exhibit strong consistency across ViT layers. In particular, tokens that exhibit large changes at early layers tend to remain salient in deeper layers. Thus, index selection is performed only in the first layer and reused across subsequent layers, significantly reducing overhead.

$$Q_t^{(\mathcal{U}_t)} = W_Q T_t^{(\mathcal{U}_t)}, \quad K_t^{(\mathcal{U}_t)} = W_K T_t^{(\mathcal{U}_t)}, \quad V_t^{(\mathcal{U}_t)} = W_V T_t^{(\mathcal{U}_t)}. \tag{4}$$

Updated tokens are recomputed, while others reuse cached key value pairs. The full representation is assembled via sparse scatter:

$$K_t^{(\text{full})} = \text{scatter}(K_t^{(\mathcal{U}_t)}, K_{\text{ref}}^{(\mathcal{R}_t)}). \tag{5}$$

Attention is computed only for updated tokens, while reused tokens directly inherit cached outputs. After residual addition, all tokens are processed by the MLP layer, ensuring full forward propagation and stable training. As illustrated in Fig. 2, the proposed mechanism selectively updates a subset of tokens while reusing the remaining ones across consecutive frames. Specifically, the first Transformer layer performs token selection based on cosine similarity, where tokens with large feature variations are identified as dynamic regions (top-ρN), while the rest are treated as temporally redundant and skipped. The selected indices are then reused across subsequent layers, where only the corresponding tokens are recomputed, and all others directly inherit cached representations. This design is motivated by the empirical observation, discussed in Sec.1, that token similarity patterns remain highly consistent across layers.

C. Efficient Flow Matching Policy with Step Compression

We analyze the intrinsic structure of the learned flow dynamics. As shown in Fig. 1(c), the singular value spectrum of the velocity sequence exhibits a rapid decay, with most of the energy concentrated in the top two components. Motivated by this observation, we model the velocity dynamics as a low rank process, where the trajectory evolves within a low dimensional subspace.

To further validate this observation, we visualize the action trajectories in the principal component space in Fig. 4. We observe that the trajectories produced by the compressed 2-step solver closely match those of the original 10-step solver in the top-2 principal components, indicating that the trajectory evolution is intrinsically low dimensional and can be captured by a small number of degrees of freedom. Use $\{v_t\}_{t=0}^{T-1}$ denote the velocity sequence. Under the low rank assumption, each velocity can be expressed as a linear combination of a small number of basis directions:

$$v_t = \sum_{k=1}^{r} \alpha_{t,k} u_k + \epsilon_t. \tag{6}$$

In practice, we choose v_0 and v_7 as anchor velocities, as they respectively represent early stage global direction and mid trajectory refinement, providing a sufficiently good basis to capture the dominant dynamics under the low rank assumption. Namely the initial velocity v_0 and a representative mid trajectory velocity v_7, leading:

$$v_t \approx a_t v_0 + b_t v_7 + \epsilon_t. \tag{7}$$

TABLE I

COMPARISON OF VLA MODELS ON THE LIBERO BENCHMARK. WE REPORT SUCCESS RATE (SR) AND EFFICIENCY METRICS. SR FOR EACH COLUMN CORRESPONDS TO INDIVIDUAL TASK CATEGORIES, WHILE MEAN SR DENOTES THE AVERAGE PERFORMANCE ACROSS ALL TASKS.

Method	LIBERO 10	Goal	Spatial	Object	Mean SR	Steps	Avg Time (ms)	FPS	TFLOPs
π_0 [4]	83.2	90.0	98.0	97.8	92.3	10	276.4	3.6	4.48
$\pi_{0.5}$ [14]	88.6	92.0	98.6	98.2	94.4	10	286.9	3.5	4.48
OpenVLA [16]	53.7	79.2	84.7	88.4	76.5	8	629.6	1.6	8.82
X-VLA [42]	88.9	91.2	98.2	98.0	94.1	10	256.8	3.9	–
Efficient VLA [34]	81.2	91.6	95.8	97.8	91.6	10	144.1	6.9	1.48
π_0 **+ Our Method**	80.2	89.6	96.8	97.2	91.0	**2**	156.2	6.4	2.86
$\pi_{0.5}$ **+ Ours Method**	88.2	91.2	97.8	98.0	93.8	**2**	**121.2**	**8.2**	1.23

Under Euler integration, the final state is given by:

$$x_T = x_0 + \sum_{t=0}^{T-1} \Delta t \cdot v_t. \tag{8}$$

Substituting the low rank approximation yields:

$$x_T \approx x_0 + \alpha v_0 + \beta v_7 + b, \tag{9}$$

where α, β aggregate temporal contributions and b absorbs residual errors. As illustrated in Fig. 3, this formulation allows us to reconstruct the full trajectory using only two representative steps, effectively reducing the multi step solver to a compact 2-step formulation. The coefficients (α, β, b) depend on the observation and are not directly available in practice. To estimate these parameters, we introduce a lightweight adaptor g_ϕ that directly predicts:

$$(\alpha, \beta, b) = g_\phi(v_0). \tag{10}$$

Training the adaptor by minimizing the discrepancy between the reconstructed trajectory using the compressed 2-step formulation and the original full step trajectory in practice overall:

$$\min_{\phi} \ \mathbb{E}_{o \sim \mathcal{D}_{\text{cal}}} \Big[\| x_T^{(2)} - x_T^{(10)} \|_2^2 \Big]. \tag{11}$$

IV. EXPERIMENT

A. Experimental Setup

a) Simulation Benchmark.: We evaluate our framework on the LIBERO and RobotWin benchmark[20, 7], these two widely used simulation suites for tabletop manipulation tasks. The model takes 224×224 RGB observations and natural language instructions as input and directly outputs joint space actions for 7-DoF manipulation. Success rate is adopted as the primary evaluation metric.

b) Real Robot Platform.: To validate the proposed acceleration framework in real world settings, we deploy our method on the **Marvin Pro** robotic platform, a bimanual robot with two 7-DoF arms. Visual observations are provided by three RGB-D cameras: a head mounted Intel RealSense D435 and two wrist cameras SHW 5G. Demonstration data are collected via teleoperation using a Meta Quest 3. The policy outputs actions in Cartesian space control.

B. Simulation Results

We evaluate our method on the LIBERO benchmark and compare it with representative VLA baselines. As shown in Table I, our approach consistently maintains competitive task performance while significantly reducing the number of sampling steps overall.

In particular, applying our method to $\pi_{0.5}$ achieves a mean success rate of 93.8%, broadly comparable to the original $\pi_{0.5}$ and X-VLA, despite significantly reducing the sampling process from 10 steps to 2. This strongly suggests that a substantial portion of iterative updates in flow based policies may be largely redundant. Meanwhile, our method substantially improves efficiency, reducing inference latency from 286.9 ms to 121.2 ms while correspondingly increasing control frequency from 3.5 FPS to 8.2 FPS. Similar trends are observed for π_0, demonstrating that the proposed step compression strategy generalizes across different policy backbones.

Compared to prior methods that primarily optimize perception or language modules, our approach directly and effectively reduces the dominant cost in action generation, thereby ultimately leading to more substantial overall end to end speedup in practice.

Moreover, the efficiency gains are achieved without introducing additional model complexity or training overhead. Overall, these results clearly demonstrate a favorable trade off between efficiency and performance, making the approach suitable for real time robotic deployment without sacrificing task performance.

C. Inference Efficiency Analysis

We analyze the source of acceleration by decomposing the inference pipeline into perception (ViT), language reasoning (LLM), and action generation (Action Expert), as summarized in Table II.

The results clearly show that, in the evaluated VLA models, the primary computational bottleneck lies in the Action Expert, which largely dominates the overall latency, while the ViT and LLM contribute a relatively much smaller portion overall.

Our method directly targets this bottleneck. While leaving the ViT and LLM latency largely unchanged, it significantly reduces the cost of the Action Expert, leading to a substantial reduction in end to end latency and a corresponding increase in control frequency.

TABLE II

INFERENCE TIME BREAKDOWN AND COMPUTATIONAL COST ON LIBERO AND ROBOTWIN. NUMBERS IN PARENTHESES INDICATE RELATIVE VALUES NORMALIZED TO $\pi_{0.5}$ AND π_0 (100.0%).

LIBERO Benchmark

Method	SR (%) ↑	ViT (ms) ↓	LLM (ms) ↓	Action Expert (ms) ↓	Total (ms) ↓	FPS ↑	TFLOPs ↓
$\pi_{0.5}$	94.4	40.1 (100%)	42.5 (100%)	212.6 (100%)	293.2 (100%)	3.4 (100%)	4.48 (100%)
ToMe[5]	75.3	31.8 (79%)	32.8 (77%)	202.3 (95%)	266.9 (91%)	3.7 (109%)	3.42 (76%)
ToFu[24]	71.6	31.2 (78%)	30.1 (71%)	204.2 (96%)	265.5 (91%)	3.8 (112%)	3.25 (73%)
V2Drop[6]	68.2	32.4 (81%)	29.1 (68%)	202.8 (95%)	264.3 (90%)	3.8 (112%)	3.18 (71%)
SnapKV[18]	89.8	27.2 (68%)	26.1 (61%)	196.3 (92%)	249.6 (85%)	4.0 (118%)	3.62 (81%)
SparseVLM[40]	92.2	30.2 (75%)	31.1 (73%)	199.6 (94%)	260.9 (89%)	3.8 (112%)	3.85 (86%)
DART[35]	91.8	24.6 (61%)	22.8 (54%)	201.7 (95%)	249.1 (85%)	4.0 (118%)	3.41 (76%)
VLA-cache [33]	90.2	28.2 (70%)	20.2 (48%)	198.2 (93%)	246.6 (84%)	4.1 (121%)	3.57 (80%)
Ours	**93.8**	**38.6 (96%)**	**42.3 (99%)**	**40.9 (19%)**	**121.8 (42%)**	**8.2 (241%)**	**1.23 (64%)**

RoboTwin 2.0 Benchmark

Method	TOP10 SR (%) ↑	ViT (ms) ↓	LLM (ms) ↓	Action Expert (ms) ↓	Total (ms) ↓	FPS ↑	TFLOPs ↓
π_0	82.2	41.25 (100%)	26.05 (100%)	226.85 (100%)	298.46 (100%)	3.35 (100%)	4.38 (100%)
ToMe[5]	66.1	37.11 (90%)	26.18 (101%)	200.60 (88%)	268.97 (90%)	3.72 (111%)	3.35 (76%)
ToFu[24]	60.4	36.50 (88%)	25.96 (100%)	199.34 (88%)	265.00 (89%)	3.77 (113%)	3.38 (77%)
V2Drop[6]	59.8	36.03 (87%)	26.15 (100%)	198.25 (87%)	264.70 (89%)	3.78 (113%)	3.60 (82%)
SnapKV[18]	78.3	34.31 (83%)	25.68 (99%)	188.30 (83%)	252.87 (85%)	3.95 (118%)	3.90 (89%)
SparseVLM[40]	76.4	36.58 (89%)	25.89 (99%)	200.29 (88%)	265.98 (89%)	3.76 (112%)	3.85 (88%)
Ours	**81.5**	**39.6 (96%)**	**25.8 (99%)**	**43.1 (19%)**	**125.4 (42%)**	**8.0 (239%)**	**2.80 (64%)**

This suggests that a major source of inefficiency originates from the multi step policy sampling process. In contrast, many existing efficiency approaches primarily focus on optimizing perception or language modules, which may lead to limited end to end gains when action generation dominates the computational cost.

Moreover, this finding highlights an important system level insight: optimizing upstream modules such as perception or language reasoning alone may yield limited overall acceleration when downstream policy execution remains the dominant bottleneck. In contrast, directly reducing the computational cost of action generation leads to more substantial end to end efficiency gains, in practice overall, emphasizing the importance of holistic optimization across the entire VLA pipeline. Overall, these results clearly highlight that policy level compression provides an effective strategy for improving system level efficiency in flow based VLA models, achieving both high efficiency and competitive performance in practice.

D. System Level Ablation

We analyze the individual contribution of each component in Table III, including temporal token reuse in the vision encoder and the proposed 2-step efficient policy. Enabling Token Reuse alone reduces the ViT latency from 40.1 ms to 28.5 ms with almost no performance degradation (93.9% vs. 94.4%), showing that substantial temporal redundancy exists in visual tokens across consecutive frames. Compared to Token Reuse, the Efficient Policy contributes the majority of the latency reduction, further confirming that action generation is the dominant source of computational cost.

Applying the Efficient Policy alone dramatically reduces the Action Expert latency from 212.6 ms to 41.5 ms, significantly lowering the total inference time from 293.2 ms to 123.9 ms while maintaining comparable task performance (94.1%). This suggests that compressing the multi step flow matching process largely maintains action performance while reducing redundant policy updates.

Combining both components yields the full model, achieving the lowest latency (111.6 ms, 9.0 FPS) with minimal performance drop (93.8%). These results validate that two components target different parts of the pipeline and provide complementary gains, enabling system level acceleration without sacrificing task success rate.

E. Real World Evaluation

We further validate our method on a Marvin Pro Robot, as shown in Fig. 5. Across six representative tasks, our 2-step policy achieves a comparable overall success rate to the original $\pi_{0.5}$ (95.4% vs. 97.2%), while consistently improving the time constrained success rate from 77.1% to 82.3% (Table IV).

This improvement clearly indicates that reducing policy latency increases the effective control frequency, allowing the robot to react more promptly to state deviations and complete long horizon tasks within the given time budget. Qualitatively

TABLE III

SYSTEM LEVEL ABLATION OF OUR METHOD. WE REPORT SUCCESS RATE AND EFFICIENCY ON LIBERO. "TOKENREUSE" ENABLES TEMPORAL TOKEN REUSE IN THE VISION ENCODER, AND "2-STEP" COMPRESSES FLOW MATCHING FROM 10 STEPS TO 2 STEPS.

Method	TokenReuse	Efficient Policy	Mean SR (%)	ViT (ms)↓	LLM (ms)↓	Action (ms)↓	Total (ms)↓ / FPS↑
Baseline ($\pi_{0.5}$, 10-step)	✗	✗	94.4	40.1	42.5	212.6	293.2 / 3.4
Token Reuse	✓	✗	93.9	28.5	42.6	211.8	282.9 / 3.5
Efficient Policy	✗	✓	94.1	40.0	42.4	41.5	123.9 / 8.1
Ours	✓	✓	**93.8**	**28.4**	**42.3**	**40.9**	**111.6 / 9.0**

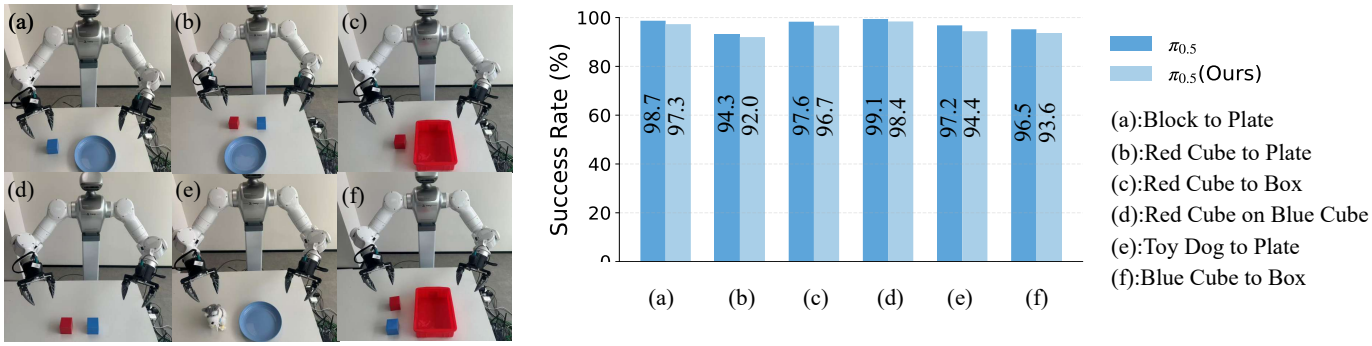

Fig. 5. **Real world manipulation performance on the Marvin Pro robot. Left:** Representative task snapshots from six manipulation tasks: (a) put the blue block on the plate, (b) place the red block onto the blue plate, (c) put the red block into the box, (d) stack the red block on top of the blue block, (e) pick up the dog toy and place it on the plate, (f) place the blue block into the red box. **Right:** Task level success rate comparison between $\pi_{0.5}$ with the standard 10-step policy and our method.

TABLE IV

REAL ROBOT EVALUATION. WE REPORT TASK SUCCESS RATE (SR) AND SUCCESS RATE WITHIN 30 SECONDS (SR@30S) ACROSS SIX MANIPULATION TASKS.

Task	$\pi_{0.5}$		$\pi_{0.5}$ (Ours)	
	SR (%)	SR@30s (%)	SR (%)	SR@30s (%)
(a) Put the block on the plate	98.7 (100%)	78.7 (79.7%)	97.3 (98.6%)	89.3 (90.5%)
(b) Place the red cube onto the plate	97.3 (100%)	79.3 (81.5%)	92.0 (94.6%)	82.0 (84.3%)
(c) Place the red cube in the box	98.3 (100%)	81.3 (82.7%)	96.7 (98.4%)	91.7 (93.3%)
(d) Place the dog toy onto the plate	99.4 (100%)	68.4 (68.8%)	98.4 (99.0%)	75.4 (75.9%)
(e) Place the red cube onto the blue cube	96.8 (100%)	66.8 (69.0%)	94.4 (97.5%)	65.4 (67.6%)
(f) Place the blue cube in the red box	95.2 (100%)	88.2 (92.6%)	93.6 (98.3%)	90.6 (95.2%)
Mean	**97.2** (100%)	77.1 (79.3%)	95.4 (98.1%)	**82.3** (84.7%)

(Fig. 5), the compressed policy consistently maintains stable and precise manipulation behavior across tasks requiring multi stage execution, without introducing additional failure modes.

Per task results suggest that the gains mainly arise from faster task completion rather than sacrificing accuracy, implying that step compression preserves the underlying action trajectory while removing redundant intermediate updates. This indicates that the essential geometric structure of the trajectory is maintained while unnecessary iterative refinements are effectively eliminated, leading to more efficient policy execution. Consistent observations from rollout trajectories show in Fig. 6 confirm 2-step policy produces smooth and goal directed behaviors under time constrained execution. The resulting trajectories remain stable and coherent across different tasks, without introducing oscillations or degraded control precision, demonstrating that the compressed policy retains reliable dynamics despite significantly reduced inference steps.

V. CONCLUSION, LIMITATIONS AND FUTURE WORK

We presented a system level acceleration framework for Vision-Language-Action models that reduces temporal redundancy in both perception and policy inference. By combining temporally aware visual token reuse with a compressed 2-step flow matching policy, the proposed method significantly improves inference efficiency while maintaining competitive manipulation performance. Experiments on the LIBERO benchmark and real robot tasks demonstrate that our approach achieves a favorable balance between task success rate and real time control efficiency. Nevertheless, several limitations remain. The compressed policy is mainly validated on short

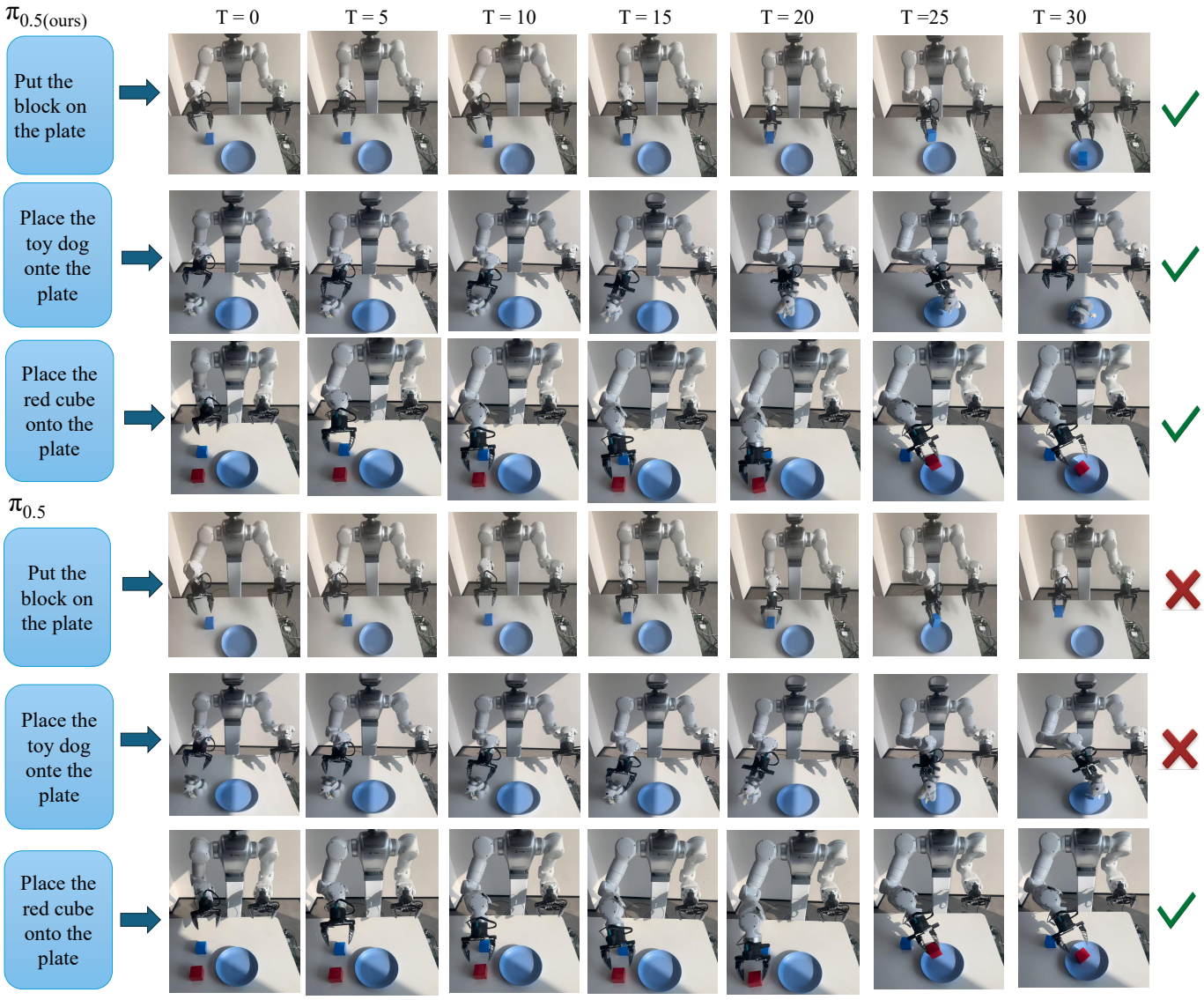

Fig. 6. **Task rollout comparison under a fixed time budget.** Each row shows a rollout trajectory from $T = 0$ to $T = 30$ for representative manipulation tasks. The top three rows correspond to $\pi_{0.5}$ with our 2-step compressed policy, while the bottom three rows show the original $\pi_{0.5}$ policy. Each column visualizes the robot state at different timestamps during execution. The results show that our compressed policy consistently completes tasks within the time budget while preserving stable manipulation behavior. The results show that our compressed policy consistently completes tasks within the time budget while preserving stable manipulation behavior.

horizon manipulation tasks, and its effectiveness for long horizon planning or highly dynamic environments requires further investigation. In future work, we will explore adaptive step scheduling strategies and extend the framework to more diverse tasks.

References

[1] Anas Awadalla, Irena Gao, Josh Gardner, Jack Hessel, Yusuf Hanafy, Wanrong Zhu, Kalyani Marathe, Yonatan Bitton, Samir Gadre, Shiori Sagawa, et al. Openflamingo: An open-source framework for training large autoregressive vision-language models. *arXiv preprint arXiv:2308.01390*, 2023.

[2] Shaojie Bai. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[3] K Black et al. pi0: A vision-language-action flow model for general robot control. *Robotics: Science and Systems XXI*, 2024.

[4] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[5] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. *arXiv preprint arXiv:2210.09461*, 2022.

[6] Junjie Chen, Xuyang Liu, Zichen Wen, Yiyu Wang, Siteng Huang, and Honggang Chen. Variation-aware vision token dropping for faster large vision-language models, 2025. URL https://arxiv.org/abs/2509.01552.

[7] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

[8] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

[9] Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 45(9):10850–10869, 2023.

[10] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[11] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint arXiv:2401.02117*, 2024.

[12] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, et al. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, 45(1):87–110, 2022.

[13] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[14] 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.

[15] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

[16] 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. *arXiv preprint arXiv:2406.09246*, 2024.

[17] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pages 12888–12900. PMLR, 2022.

[18] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

[19] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

[20] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

[21] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[22] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *Machine Intelligence Research*, pages 1–22, 2025.

[23] Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. A survey on vision-language-action models for embodied ai. *arXiv preprint arXiv:2405.14093*, 2024.

[24] Pratyush Maini, Zhili Feng, Avi Schwarzschild, Zachary C Lipton, and J Zico Kolter. Tofu: A task of fictitious unlearning for llms. *arXiv preprint arXiv:2401.06121*, 2024.

[25] Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave

Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, et al. Imitating human behaviour with diffusion models. *arXiv preprint arXiv:2301.10677*, 2023.

[26] 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.

[27] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[28] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

[29] Ranjan Sapkota, Yang Cao, Konstantinos I Roumeliotis, and Manoj Karkee. Vision-language-action models: Concepts, progress, applications and challenges. *arXiv preprint arXiv:2505.04769*, 2025.

[30] 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.

[31] Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.

[32] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

[33] Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Towards efficient vision-language-action model via adaptive token caching in robotic manipulation. *arXiv e-prints*, pages arXiv–2502, 2025.

[34] 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.

[35] Shicheng Yin, Kaixuan Yin, Yang Liu, Weixing Chen, and Liang Lin. Dart: Differentiable dynamic adaptive region tokenizer for vision foundation models. *arXiv preprint arXiv:2506.10390*, 2025.

[36] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. *Neural computation*, 31 (7):1235–1270, 2019.

[37] Yang Yue, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang. Deervla: Dynamic inference of multimodal large language models for efficient robot execution. *Advances in Neural Information Processing Systems*, 37:56619–56643, 2024.

[38] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11975–11986, 2023.

[39] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[40] 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*, 2024.

[41] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[42] Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, et al. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. *arXiv preprint arXiv:2510.10274*, 2025.

[43] 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 *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

APPENDIX A
IMPLEMENTATION DETAILS

a) Hardware Setup.: All simulation experiments are conducted on a workstation equipped with an NVIDIA A100 GPU, while real-world deployment runs on a workstation equipped with an NVIDIA RTX 4090 GPU. Table V summarizes the hardware configurations used for simulation and real-world inference.

TABLE V
HARDWARE CONFIGURATION FOR SIMULATION AND REAL-WORLD INFERENCE.

Component	Simulation	Real Robot
GPU	NVIDIA A100	NVIDIA RTX 4090
CPU	Intel Xeon Platinum 8368	AMD Ryzen 9 9950X3D
Memory	80 GB	48 GB
Framework	PyTorch	PyTorch
Precision	FP16	FP16

APPENDIX B
EFFICIENCY ANALYSIS

We further analyze the efficiency of our framework from two aspects: (1) reducing the number of flow-matching solver steps in the policy network, and (2) applying temporal token reuse in the visual encoder.

a) Effect of Flow Matching Solver Steps.: We first analyze the inference speedup obtained by reducing the number of flow matching solver steps. Let N denote the number of solver steps used during inference. Since each step requires a forward pass through the policy network, the inference latency approximately scales linearly with N.

Table VI reports the latency and relative speedup under different solver step configurations. The baseline corresponds to the original policy using $N = 10$ steps.

TABLE VI
INFERENCE LATENCY AND SPEEDUP UNDER DIFFERENT SOLVER STEP NUMBERS N.

Steps (N)	Latency (ms)	Speedup
10 (baseline)	286.9	1.0×
8	230.1	1.25×
6	178.5	1.61×
4	145.7	1.97×
2 (ours)	121.2	2.37×

As expected, reducing the solver steps leads to substantial latency reduction. However, naive step reduction often causes noticeable degradation in task success rate due to inaccurate trajectory integration. Our proposed 2-step compressed solver preserves the dominant flow trajectory through the low-rank velocity approximation described in Sec. 3.3, enabling aggressive step reduction while maintaining stable manipulation performance.

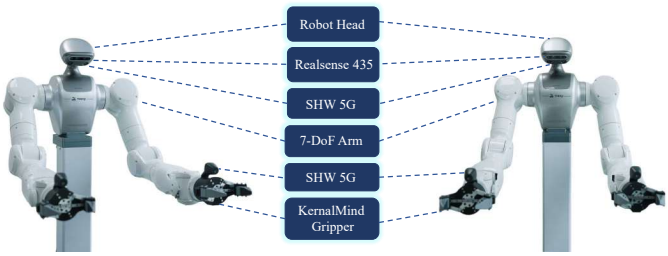

Fig. 7. **Real robot platform.** Dual-arm robot with 7-DoF manipulators, SHW 5G wrist modules, and KernalMind grippers. A RealSense 435 camera is mounted on the head for visual input.

b) Effect of Temporal Token Reuse Ratios.: We further analyze the effect of different token reuse ratios in the visual encoder. Let r denote the proportion of tokens that are recomputed at each timestep, while the remaining tokens are reused from the previous frame.

Table VII reports the task success rate under different token update ratios. We observe that reducing the token update ratio slightly affects perception fidelity, which may lead to a gradual decrease in task success rate. However, moderate reuse ratios still maintain strong performance, indicating that most visual tokens remain temporally stable across adjacent frames.

In practice, we find that updating approximately 40% of the tokens provides a good balance between visual representation accuracy and computational efficiency, and this configuration is adopted in our final system.

TABLE VII
EFFECT OF DIFFERENT TOKEN UPDATE RATIOS r ON TASK SUCCESS RATE.

Update Ratio (r)	Mean SR (%)
1.0 (baseline)	94.4
0.2	94.1
0.3	94.0
0.4	93.8
0.5	92.6
0.6	91.8

APPENDIX C
FULL ROBOTWIN BENCHMARK RESULTS.

We further report detailed evaluation results on the RoboTwin benchmark, which contains a diverse set of manipulation tasks covering object relocation, stacking, and interaction scenarios. As shown in Table VIII, our method achieves competitive success rates across a wide range of RoboTwin tasks, maintaining performance comparable to the baseline while outperforming other acceleration methods.

In addition, Table IX demonstrates that our approach significantly reduces inference latency, achieving more than 2× speedup compared to the base model.

Task	Base	Ours	ToMe	ToFu	V2Drop	SnapKV	SparseVLM
adjust_bottle	85	72	70	64	62	81	81
beat_block_hammer	76	74	60	60	2	18	60
blocks_ranking_rgb	46	36	40	32	10	0	36
blocks_ranking_size	6	2	12	14	10	0	13
click_alarmclock	68	48	52	48	46	64	64
click_bell	40	34	38	38	18	30	38
dump_bin_bigbin	84	88	68	62	62	80	80
grab_roller	92	94	76	70	68	88	88
handover_block	50	52	16	6	30	0	11
handover_mic	87	92	68	62	64	83	83
hanging_mug	6	10	0	0	0	0	0
lift_pot	52	68	46	44	10	38	45
move_can_pot	60	56	32	16	16	10	24
move_pillbottle_pad	32	30	20	20	12	14	20
move_playingcard_away	52	28	24	16	12	20	20
move_stapler_pad	0	0	0	0	0	0	0
open_laptop	80	36	50	56	42	28	53
open_microwave	72	66	78	78	26	0	78
pick_diverse_bottles	40	56	38	38	6	6	38
pick_dual_bottles	58	70	58	54	28	12	56
place_a2b_left	26	18	20	16	14	8	18
place_a2b_right	10	10	14	16	2	4	15
place_bread_basket	38	32	34	34	22	16	34
place_bread_skillet	26	40	24	22	12	10	23
place_burger_fries	68	80	55	50	46	64	65
place_can_basket	50	48	50	48	32	42	49
place_cans_plasticbox	30	16	10	16	20	0	13
place_container_plate	90	89	72	66	66	86	86
place_dual_shoes	28	22	20	14	6	0	17
place_empty_cup	60	56	46	42	44	57	57
place_fan	10	0	0	0	0	8	0
place_mouse_pad	8	6	0	0	0	0	0
place_object_basket	24	48	20	8	2	0	14
place_object_scale	10	12	10	6	0	4	8
place_object_stand	68	58	58	36	32	24	47
place_phone_stand	30	32	26	28	8	4	27
place_shoe	48	46	40	38	22	18	39
press_stapler	44	58	20	16	20	28	18
put_bottles_dustbin	16	24	8	0	0	0	4
put_object_cabinet	30	22	18	16	4	0	17
rotate_qrcode	56	56	50	36	22	4	43
scan_object	22	22	20	8	6	10	14
shake_bottle_horizontally	96	100	80	72	72	92	92
shake_bottle	92	96	74	68	68	88	88
stack_blocks_three	32	32	38	28	0	10	33
stack_blocks_two	62	62	48	46	42	28	47
stack_bowls_three	44	56	16	8	0	6	12
stack_bowls_two	90	90	68	70	50	36	39
stamp_seal	16	36	18	20	2	16	19
turn_switch	24	26	22	8	0	20	15
Average	46.68	46.10	36.50	32.28	22.76	25.10	36.82

TABLE VIII

SUCCESS RATE COMPARISON ACROSS TASKS UNDER DIFFERENT ACCELERATION METHODS.

Task	Base	Ours	ToMe	ToFu	V2Drop	SnapKV	SparseVLM
adjust_bottle	295.4	124.1	266.2	262.3	261.9	250.3	263.2
beat_block_hammer	313.1	132.1	283.3	279.3	278.9	266.9	280.1
blocks_ranking_rgb	321.5	135.1	290.8	286.7	286.3	274.1	287.6
blocks_ranking_size	321.2	134.3	290.6	286.4	286.1	273.8	287.3
click_alarmclock	285.6	120.0	257.4	253.5	253.2	241.9	254.4
click_bell	295.1	124.0	267.0	263.2	262.9	251.6	264.1
dump_bin_bigbin	305.8	128.5	275.6	271.5	271.2	259.0	272.4
grab_roller	298.2	125.3	268.7	264.7	264.4	252.6	265.6
handover_block	307.5	129.4	278.2	274.2	273.9	262.2	275.1
handover_mic	299.1	125.7	269.5	265.5	265.2	253.4	266.4
hanging_mug	319.4	133.8	288.9	284.8	284.5	272.3	285.7
lift_pot	312.2	131.0	282.4	278.5	278.1	266.2	279.3
move_can_pot	308.8	129.7	279.4	275.4	275.1	263.3	276.2
move_pillbottle_pad	307.4	128.7	278.1	274.1	273.8	262.1	275.0
move_playingcard_away	304.2	127.5	275.2	271.3	271.0	259.4	272.2
move_stapler_pad	307.1	128.6	277.8	273.9	273.6	261.8	274.7
open_laptop	315.5	132.8	285.4	281.3	281.0	268.9	282.2
open_microwave	317.7	133.1	287.4	283.3	283.0	270.8	284.2
pick_diverse_bottles	313.2	131.7	283.4	279.4	279.0	267.0	280.2
pick_dual_bottles	314.5	131.5	284.5	280.5	280.1	268.1	281.3
place_a2b_left	305.2	128.3	276.1	272.2	271.9	260.2	273.1
place_a2b_right	305.5	127.7	276.4	272.5	272.2	260.5	273.3
place_bread_basket	309.8	130.0	280.3	276.3	276.0	264.1	277.1
place_bread_skillet	311.2	131.1	281.5	277.5	277.2	265.3	278.4
place_burger_fries	312.4	131.3	281.5	277.4	277.0	264.6	278.3
place_can_basket	308.2	129.5	278.8	274.9	274.6	262.8	275.7
place_cans_plasticbox	310.5	130.3	280.9	276.9	276.6	264.7	277.8
place_container_plate	301.6	126.7	271.8	267.8	267.5	255.5	268.6
place_dual_shoes	316.2	133.4	286.0	282.0	281.7	269.5	282.8
place_empty_cup	292.4	122.8	263.5	259.6	259.3	247.7	260.5
place_fan	315.8	132.2	285.7	281.6	281.3	269.2	282.5
place_mouse_pad	302.5	126.5	273.7	269.8	269.5	257.9	270.6
place_object_basket	309.7	130.6	280.2	276.2	275.9	264.0	277.1
place_object_scale	311.4	131.0	281.7	277.7	277.4	265.5	278.6
place_object_stand	311.9	131.4	282.2	278.1	277.8	265.9	279.0
place_phone_stand	306.5	128.3	277.3	273.3	273.1	261.3	274.2
place_shoe	305.8	128.4	276.7	272.7	272.4	260.7	273.6
press_stapler	301.1	126.8	272.4	268.5	268.2	256.7	269.4
put_bottles_dustbin	318.2	134.3	287.8	283.7	283.4	271.2	284.6
put_object_cabinet	320.5	135.2	289.9	285.8	285.5	273.2	286.7
rotate_qrcode	299.4	125.2	270.9	267.0	266.7	255.3	267.9
scan_object	300.2	126.4	271.6	267.7	267.4	256.0	268.6
shake_bottle_horizontally	297.4	125.0	268.0	264.0	263.7	251.9	264.9
shake_bottle	296.8	124.6	267.5	263.5	263.2	251.4	264.4
stack_blocks_three	326.6	137.8	295.4	291.2	290.8	278.3	292.1
stack_blocks_two	321.4	135.1	290.7	286.6	286.3	273.9	287.5
stack_bowls_three	329.2	138.0	297.8	293.5	293.2	280.5	294.5
stack_bowls_two	325.1	136.2	294.1	289.9	289.5	277.1	290.8
stamp_seal	302.8	127.7	274.0	270.0	269.7	258.2	270.9
turn_switch	297.4	125.0	269.1	265.2	264.9	253.6	266.1
Avg	309.30	129.67	278.37	274.28	273.95	262.86	275.32

TABLE IX

AVERAGE INFERENCE LATENCY (MS) ACROSS ROBOTWIN TASKS UNDER DIFFERENT ACCELERATION METHODS.