VADP-VLA: VIEW-ADAPTIVE DYNAMIC VISUAL TOKEN PRUNING FOR EFFICIENT VLA INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) models generate robot actions from visual observations, language instructions, and robot states, but the large number of visual tokens produced by multiple cameras increases inference cost. Existing methods reduce visual computation by exploiting visual saliency, temporal redundancy, or action relevance. Because camera views differ in both content and temporal dynamics, however, the validity of historical information and the appropriate visual token budget must be assessed separately for each view. We propose VADP-VLA, a training-free framework for view-adaptive dynamic visual token pruning. It estimates language-conditioned visual token saliency within each view and transfers historical saliency through a per-view keyframe cache. Agreement between current and historical saliency controls cache refresh and token retention independently for each view. Tokens that are not retained are removed before later language-model layers, shortening the sequence processed by subsequent computation. Across the four LIBERO suites, VADP-VLA achieves an average success rate of 95.25%, compared with 94.55% for the unpruned OpenVLA-OFT baseline, while reducing normalized language-model FLOPs to 41% of the baseline and model-internal latency from 73.12 ms to 48.89 ms, a $1.50\times$ speedup. In LIBERO experiments with $\pi_{0.5}$, average success changes from 96.30% with the unpruned $\pi_{0.5}$ baseline to 94.65%, normalized language-model FLOPs fall to 45%, and model-internal latency decreases from 37.38 ms to 25.43 ms. Across four real-robot tasks, each method is evaluated in 80 trials: VADP-VLA succeeds in $63/80$ trials versus $67/80$ for the unpruned $\pi_{0.5}$ baseline, while VLA model-call latency falls from 59.93 ms to 42.58 ms, approximately a $1.41\times$ speedup. These results show reduced VLA inference cost in the evaluated settings, alongside a modest decrease in real-robot task success relative to the unpruned baseline.

1 INTRODUCTION

Vision-language-action (VLA) policies predict executable robot actions from visual observations, language instructions, and robot states. RT-2, OpenVLA, and Octo illustrate different ways to transfer large-scale vision-language pretraining to robot control (Brohan et al., 2023; Kim et al., 2024; Octo Model Team et al., 2024). VLA policies, however, typically include computationally demanding visual encoders and multimodal backbones. In a closed-loop task, the robot must acquire a new observation and invoke the policy again after executing predicted actions. Action chunking reduces the number of policy calls by predicting multiple consecutive actions in one forward pass, yet completing a manipulation task generally still requires repeated inference (Kim et al., 2025). Reducing the computation of each call is therefore important for efficient VLA inference. Multiple cameras further increase visual computation. Each encoded and projected image produces a set of visual tokens, and combining the views lengthens the sequence processed by subsequent multimodal modules. Because visual tokens differ in their relevance to the current task, less relevant tokens can be removed to reduce later computation. Effective visual token pruning nevertheless faces three challenges. First, token importance should reflect the language instruction and manipulation goal rather than visual prominence alone. Second, consecutive policy calls may contain similar content, but arm motion, object movement, occlusion, and contact can invalidate historical information. Third, cameras cover different regions and scales, so their observations need not change in synchrony. Both the continued usefulness of historical information and the number of tokens to retain should there-

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.

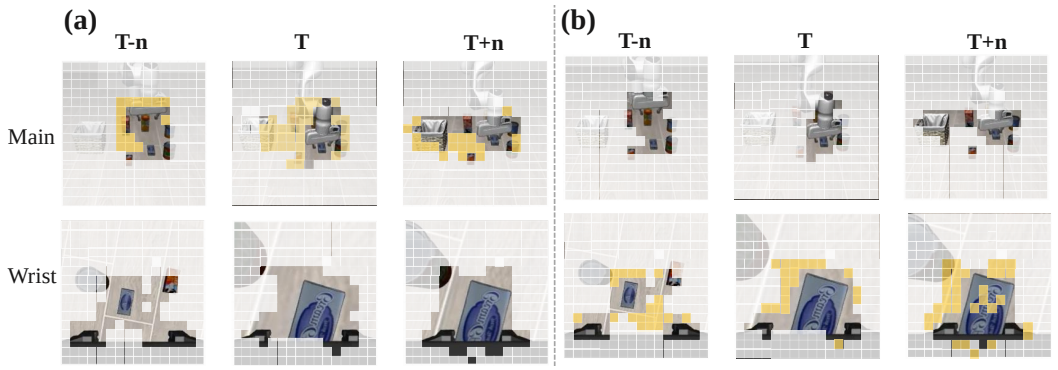


Figure 1: Illustration of token selection by conventional visual token pruning and our method. The global view is shown above the wrist view. Translucent patches indicate discarded visual tokens, and yellow outlines mark locations selected differently by the two methods. (a) An example of conventional pruning. (b) VADP-VLA selects visual tokens separately within each view. The figure illustrates selection outcomes only and does not imply that all conventional methods use the same cache or budget mechanism.

fore depend on the state of each view. Existing visual token compression methods use shallow-layer attention, text-conditioned relevance, feature diversity, or token merging to reduce subsequent computation (Chen et al., 2024; Zhang et al., 2025; Yang et al., 2025; Shang et al., 2025). More recent VLA acceleration methods incorporate temporal information, action relevance, or intermediate network signals (Liu et al., 2025; Wang et al., 2026; Ma et al., 2026). These methods reduce visual computation in different ways, but preserving camera-view identity throughout historical matching, cache management, and token budget allocation remains less explored. Sharing historical state or control decisions across views may retain unnecessary computation in a stable view while discarding task-relevant evidence from a changing view. Figure 1 illustrates the difference between an example of conventional visual token pruning and our view-aware selection. The illustration does not imply that all previous pruning methods operate identically. To address these challenges, we propose VADP-VLA, a training-free framework for view-adaptive dynamic visual token pruning in VLA inference. After the first ℓ language-model layers, it estimates the task relevance of visual tokens under the language condition. Within each camera view, current tokens are matched to tokens in a historical keyframe to transfer saliency. Agreement between current and historical saliency then governs each view's keyframe cache and the number of visual tokens used in later computation. Cache management updates the historical reference; token budgeting controls the sequence length processed by later layers. Unselected visual tokens are physically removed before those layers, while non-visual tokens remain unchanged. Our main contributions are as follows:

- We introduce per-view historical saliency modeling. The mechanism matches current observations to historical keyframes within the same camera view, transfers task-conditioned saliency, and maintains a separate keyframe cache for each view, avoiding direct transfer of historical information across cameras.

- We propose a view-adaptive dynamic token budget. Each view uses agreement between current and historical saliency to determine its cache state and retained token count. Physically removing unselected tokens shortens the visual sequence processed by subsequent language-model modules.

- We evaluate the success–efficiency trade-off on LIBERO and real-robot tasks. On LIBERO, VADP-VLA achieves 95.25% average success with OpenVLA-OFT (versus 94.55% for the unpruned baseline) at 41% normalized language-model FLOPs, and 94.65% with $\pi_{0.5}$ (versus 96.30% for the unpruned baseline) at 45% normalized FLOPs. Across four real-robot tasks, it succeeds in $63/80$ trials versus $67/80$ for the unpruned $\pi_{0.5}$ baseline while achieving a $1.41\times$ VLA model-call speedup.

2 Related Work

VLA policies and action generation. RT-2 represents actions as discrete tokens (Brohan et al., 2023), whereas OpenVLA-OFT adapts OpenVLA to continuous regression, parallel decoding, and action chunking (Kim et al., 2024; 2025). Octo uses diffusion-based action generation, while the π_0 family employs flow matching (Octo Model Team et al., 2024; Black et al., 2025; Physical Intelligence et al., 2025; Lipman et al., 2022). VADP-VLA reduces visual-token computation within each policy call without changing the policy's action interface.

Visual token compression and efficient VLA inference. FastV prunes low-impact tokens using shallow-layer attention (Chen et al., 2024); SparseVLM applies text-guided sparsification (Zhang et al., 2025); VisionZip selects informative encoder outputs (Yang et al., 2025); and LLaVA-PruMerge combines pruning and merging (Shang et al., 2025). LightVLA learns a differentiable task-oriented selector (Jiang et al., 2025). VLA-specific methods also use temporal or action signals: VLA-Pruner combines semantic saliency with temporally smoothed action relevance (Liu et al., 2025), SpecPrune-VLA controls pruning at the action and network levels (Wang et al., 2026), and SAFE-Pruner predicts importance in deeper layers (Ma et al., 2026). VLA-Cache reuses representations across policy calls (Xu et al., 2025), while DeeR-VLA adaptively changes executed network depth through early exit (Yue et al., 2024). VADP-VLA instead uses camera identity as an explicit control variable in matching, refresh, and budget allocation.

Multi-view robotic perception. Global and wrist cameras provide complementary global and local visual evidence. PerAct aggregates multi-view RGB-D observations into a voxel representation (Shridhar et al., 2022), whereas RVT reasons over multiple virtual views (Goyal et al., 2023). World-to-Wrist emphasizes the different roles of global and wrist observations in fine manipulation (Pan et al., 2026), and Multi-View Unified Camera Fields studies geometric correspondences in RGB-only multi-camera policies (Yang et al., 2026). These works focus on representation learning and fusion; we study how view-specific temporal state controls computation during closed-loop inference.

3 Motivation

In multi-camera VLA inference, views are typically acquired simultaneously, but simultaneous acquisition does not imply synchronized changes in their content or task roles. The global view covers the workspace and object layout, which may change with object displacement, arm occlusion, or scene state. The wrist view is closer to local interactions and may change with arm motion, contact, and occlusion, yet it may also remain relatively stable during parts of execution. Camera type alone therefore cannot determine which view is always more stable or dynamic. At each inference step, we must assess whether each view's historical information remains useful and whether its effect on action prediction has changed. Figure 2(a) shows changes in global and wrist observations along one task trajectory. Their pixel changes follow different trends across inference steps, indicating that their visual content does not always change together. Pixel differences reflect the magnitude of observation changes, however, not the importance of task-relevant information. Figure 2(c) further examines how each view affects action prediction by updating either the global or wrist view to its current observation while holding the other fixed, then measuring the change in the predicted action chunk. The influence of each view changes during task execution. Using the same historical state or a uniform compute budget for both views at every inference step may therefore fail to account for these time-varying differences. To represent these differences during inference, we use agreement between current and within-view historical saliency as a proxy for the validity of historical information. Figure 2(b) shows that saliency agreement follows different trajectories for the two views, indicating that this proxy can characterize their states separately. Because the agreement is computed from our saliency estimator and historical matching procedure, it serves only as a decision signal for cache management and token budgeting; it is not a direct measure of a view's true update requirement. These observations motivate three related design choices. First, current visual tokens are matched only to historical tokens from the same camera view, preventing cross-view saliency transfer. Second, each view maintains its own keyframe cache and updates historical information according to its own agreement state. Third, cache management and token budgeting have distinct roles: the former updates the historical reference, while the latter determines how many visual to-

kens participate in later computation. Together, these choices allow VADP-VLA to manage history and adjust token counts based on each view's state at the current inference step. Section 4.3 specifies the method, and Section 5.5 compares shared and independent refresh to assess per-view control.

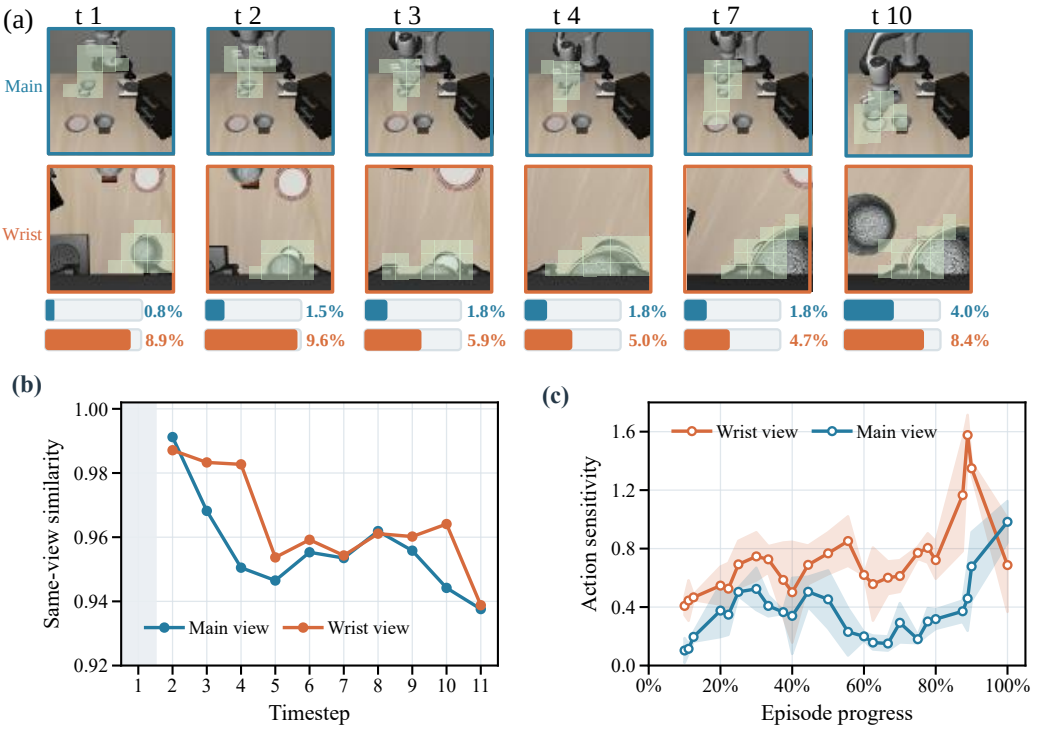


Figure 2: **Visual changes and action sensitivity across views.** (a) Global and wrist observations from consecutive policy calls. Bars show the mean absolute RGB difference from the preceding observation. (b) Within-view saliency agreement ρ_t^v with each view's latest keyframe. (c) Action-chunk sensitivity when one view is updated to the current observation while the other retains its previous observation. Curves show the mean, and shaded bands indicate $\pm$ one sample standard deviation.

4 METHOD

4.1 PROBLEM SETUP AND OVERVIEW

VADP-VLA targets repeated inference with multi-view VLA policies. It reduces per-call computation by decreasing the number of visual tokens processed by later language-model layers. The central idea is to maintain historical saliency separately for each camera view and use agreement between the current observation and its history to decide cache refresh and token budgets independently. At time step t, the model receives a language instruction x, proprioceptive state p_t, and synchronized images from a main camera $\mathcal{I}_t^m$ and a wrist camera $\mathcal{I}_t^w$. The superscripts m and w denote the global and wrist views, respectively. We use $v \in \{m, w\}$ for either view. The visual encoder and multimodal projector convert each view into a visual token matrix:

$$\mathbf{E}_t^v = \begin{bmatrix} (\mathbf{e}_{t,1}^v)^\top \\ \vdots \\ (\mathbf{e}_{t,N_v}^v)^\top \end{bmatrix} \in \mathbb{R}^{N_v \times d}, \tag{1}$$

Here, $\mathbf{e}_{t,i}^v \in \mathbb{R}^d$ is the ith projected visual token, N_v is the number of visual tokens in that view, and d is the language-model hidden dimension. In OpenVLA-OFT, each view contains 256 visual tokens, giving $N_m = N_w = 256$ and 512 tokens in total (Kim et al., 2025). Each policy call outputs a continuous eight-step action chunk with seven dimensions per step. Figure 3 provides an overview

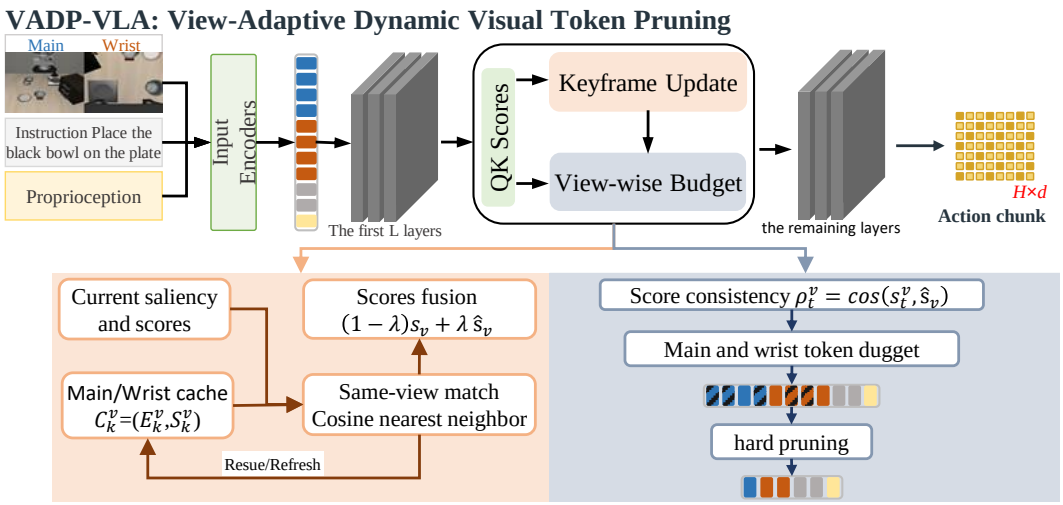


Figure 3: Overview of VADP-VLA inference. After model layer ℓ, saliency is estimated and current tokens are matched to cached tokens within the same view to transfer historical saliency. The global and wrist views independently decide cache refresh and token retention. Unselected visual tokens are then physically removed.

of VADP-VLA. The first ℓ model layers run without pruning. After layer ℓ, visual token saliency is estimated and compared with cached historical saliency to decide whether each view should refresh its keyframe. On refresh, the complete current projected visual tokens and their saliency are cached, while 20% of visual tokens are pruned before later layers. Otherwise, each view's token budget is determined from saliency agreement and pruning is applied accordingly. The first policy call initializes both caches and retains all visual tokens.

4.2 Task-Conditioned Visual Saliency Estimation

We estimate visual saliency after model layer ℓ so that token selection can use task-conditioned hidden states. Specifically, text and visual hidden states from layer ℓ are passed through the input normalization and query/key projections of layer $\ell+1$ to obtain text queries and visual keys. An ablation of query sources appears in the appendix. Let N_h be the number of attention heads, L_Q the number of text queries, and $d_h = d/N_h$ the dimension of each head. We write $\mathbf{q}_{t,h,a} \in \mathbb{R}^{d_h}$ for text query a in head h and $\mathbf{k}^v_{t,h,i} \in \mathbb{R}^{d_h}$ for visual key i of view v in that head. Averaging over text queries and attention heads gives the raw relevance of a visual token:

$$r^v_{t,i} = \frac{1}{N_h L_Q} \sum_{h=1}^{N_h} \sum_{a=1}^{L_Q} \frac{\langle \mathbf{q}_{t,h,a}, \mathbf{k}^v_{t,h,i} \rangle}{\sqrt{d_h}}, \qquad (2)$$

$$\mathbf{s}^v_t = \mathcal{N}(\mathbf{r}^v_t), \qquad \mathbf{r}^v_t = [r^v_{t,1}, \ldots, r^v_{t,N_v}]^\top.$$

Here, $\langle \cdot, \cdot \rangle$ denotes an inner product, and $\mathbf{s}^v_t \in \mathbb{R}^{N_v}$ is the current saliency vector. The normalization operator $\mathcal{N}$ is applied separately within each view. For a score vector $\mathbf{z}$, its ith output is $\mathcal{N}(\mathbf{z})_i = (z_i - \min_j z_j)/(\max_j z_j - \min_j z_j + \epsilon)$, where $\epsilon = 10^{-6}$ prevents division by zero. This probe uses raw query–key relevance without applying an additional softmax, rotary positional embedding, or attention mask. It reuses existing projection parameters and requires no additional training. The resulting $\mathbf{s}^v_t$ is used for historical agreement tests and visual token selection.

4.3 Historical Saliency Transfer and Independent Cache Refresh

Saliency from a single observation may fluctuate with occlusion or local appearance changes, motivating the use of historical information. Each view independently maintains a keyframe cache $\mathcal{C}^v_k = (\mathbf{E}^v_k, \mathbf{s}^v_k)$. Here, k is the time step of the most recently refreshed keyframe for view v before the current call. For non-initial calls, $k < t$. The global and wrist views maintain separate keyframes, so their values of k may differ. $\mathbf{E}^v_k \in \mathbb{R}^{N_v \times d}$ is the complete visual token matrix produced by the visual encoder and multimodal projector for the keyframe, where N_v is the view's

token count and d is the language-model hidden dimension. $\mathbf{s}_k^v \in \mathbb{R}^{N_v}$ contains that keyframe's task-conditioned saliency scores. Its element $s_{k,j}^v$ corresponds to visual token $\mathbf{e}_{k,j}^v$ in $\mathbf{E}_k^v$. Thus, the cached $\mathbf{E}_k^v$ supports feature matching, while $\mathbf{s}_k^v$ provides historical saliency.

Within-view historical transfer. For current visual token i, we find its most similar token by cosine similarity in the same view's keyframe and transfer that token's saliency score:

$$\begin{aligned} j_{t,i}^{v*} &= \arg\max_{1\le j\le N_v} \cos\left(\mathbf{e}_{t,i}^v, \mathbf{e}_{k,j}^v\right), \\ \widetilde{s}_{t,i}^v &= s_{k,j_{t,i}^{v*}}^v. \end{aligned} \tag{3}$$

Here, $\cos(\cdot,\cdot)$ is the inner product divided by the product of the two Euclidean norms, $j_{t,i}^{v*}$ is the matched keyframe token index, and $\widetilde{s}_{t,i}^v$ is the transferred historical saliency. The transferred scores form $\widetilde{\mathbf{s}}_t^v \in \mathbb{R}^{N_v}$. Both matching and score transfer remain within the same view.

Refresh decision and selection scores. The first call sets the current frame as the keyframe and initializes the cache. On later calls, cosine similarity between current and transferred historical saliency measures the usefulness of history and determines refresh:

$$\rho_t^v = \cos(\mathbf{s}_t^v, \widetilde{\mathbf{s}}_t^v), \qquad g_t^v = \begin{cases} 1, & \rho_t^v < \gamma_v, \\ 0, & \rho_t^v \ge \gamma_v, \end{cases} \qquad t > 1. \tag{4}$$

Here, ρ_t^v is saliency agreement and γ_v is the refresh threshold for view v; $g_t^v = 1$ means refresh, whereas $g_t^v = 0$ retains the existing cache. Low agreement triggers refresh: the current frame becomes the new keyframe, the cache becomes $(\mathbf{E}_t^v, \mathbf{s}_t^v)$, and its keyframe index is set to $k = t$. Otherwise, the keyframe index and cache remain unchanged. When agreement is high, current and historical saliency are fused. The resulting token-selection score is

$$\mathbf{s}_t^{v,\mathrm{sel}} = \begin{cases} \mathbf{s}_t^v, & g_t^v = 1, \\ \mathcal{N}((1-\lambda)\mathbf{s}_t^v + \lambda\widetilde{\mathbf{s}}_t^v), & g_t^v = 0, \end{cases}$$

where $\lambda \in [0,1]$ controls the weight of historical saliency. This rule uses history to assist selection when it remains valid and selects from the current observation when it does not. The caches are cleared at the start of each episode. The first call, $t = 1$, performs no historical matching or agreement computation: it retains all visual tokens and initializes both caches with current embeddings and saliency. Thereafter, the views decide refresh independently. Agreement and budget decisions use the pre-update cache. A refresh stores the complete projected visual tokens and current saliency, regardless of the token count retained on that call.

4.4 Update-Driven Per-View Budgets and Sequence Compression

Cache refresh determines whether historical information is replaced; the token budget determines how much visual information later modules process. VADP-VLA controls them separately: even a refreshed view has a retention budget, and each view adjusts its budget according to its own saliency agreement. For $t > 1$, thresholds $1 \ge \tau_1 > \tau_2 > \tau_3 \ge 0$ divide saliency agreement into states I–IV. The non-refresh budget mapping for view $v \in \{m, w\}$ is

$$\mathcal{B}_v(\rho) = \begin{cases} q_{\mathrm{I}}^v, & \rho \ge \tau_1, \\ q_{\mathrm{II}}^v, & \tau_2 \le \rho < \tau_1, \\ q_{\mathrm{III}}^v, & \tau_3 \le \rho < \tau_2, \\ q_{\mathrm{IV}}^v, & \rho < \tau_3. \end{cases} \tag{5}$$

The four wrist-view budget values are $(q_{\mathrm{I}}^w, q_{\mathrm{II}}^w, q_{\mathrm{III}}^w, q_{\mathrm{IV}}^w) = (0.2, 0.4, 0.6, 0.8)$. The main view maps its four states to two budget values: $(q_{\mathrm{I}}^m, q_{\mathrm{II}}^m, q_{\mathrm{III}}^m, q_{\mathrm{IV}}^m) = (0.1, 0.1, 0.8, 0.8)$. Cache refresh takes precedence over the state budget: either view retains 80% of its visual tokens on refresh and uses its view-specific budget mapping only otherwise. Thus,

$$q_t^v = \begin{cases} 0.8, & g_t^v = 1, \\ \mathcal{B}_v(\rho_t^v), & g_t^v = 0, \end{cases} \qquad t > 1. \tag{6}$$

The refresh threshold γ_v determines whether to replace the cache, whereas the state thresholds τ_1, τ_2, τ_3 determine the token budget only when no refresh occurs. Let k_t^v denote the retained token count and $\mathcal{K}_t^v \subseteq \{1, \ldots, N_v\}$ the retained within-view index set. For non-initial calls,

$$k_t^v = \mathrm{round}(q_t^v N_v), \qquad \mathcal{K}_t^v = \mathrm{TopK}\left(\mathbf{s}_t^{v,\mathrm{sel}}, k_t^v\right), \qquad t > 1, \tag{7}$$

Here, round rounds to the nearest integer, and TopK returns the indices of the k_t^v highest-scoring tokens. On the first call, $k_1^v = N_v$ and $\mathcal{K}_1^v = \{1, \ldots, N_v\}$. The maximum retention ratio after initialization is 0.8, corresponding to approximately 20% minimum pruning; actual counts are rounded as above. Finally, within-view indices in $\mathcal{K}_t^m$ and $\mathcal{K}_t^w$ are mapped back to the original multimodal sequence, and selected visual tokens retain their original order. Unselected visual tokens are physically removed before the next layer; retained tokens keep their original position indices, and the attention mask is updated consistently. The complete per-view inference procedure is provided in Appendix A (Algorithm 1).

5 Experiments

5.1 Evaluation Setup

We evaluate task success and inference efficiency on LIBERO and a real robot. The reported metrics are closed-loop success rate, normalized language-model FLOPs, and measured inference latency. LIBERO experiments run on a Linux workstation with one NVIDIA RTX 5090 32 GB GPU. FLOPs are normalized to 100% against the corresponding unpruned baseline within each base-model group; normalized FLOPs are not compared directly across base models. LIBERO latency covers the interval from language-model input to action-chunk output, whereas real-robot latency covers a VLA model call. LIBERO contains language-conditioned manipulation suites that vary spatial relations, object categories, goals, and task horizon (Liu et al., 2023). We evaluate LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long, with Long denoting the long-horizon task subset of LIBERO-100. Each suite contains 10 tasks with 50 episodes per task, or 500 episodes per suite and 2,000 across all four suites. OpenVLA-OFT receives a fixed third-person image, a wrist image, proprioceptive information, and a language instruction, then predicts and executes eight-action chunks (Kim et al., 2025). We also evaluate the unpruned $\pi_{0.5}$ baseline and $\pi_{0.5}$ with VADP-VLA on the same four suites, with 500 episodes per suite for each configuration. For real-robot evaluation, we use a fixed Seeed Studio Arm B601-DM arm equipped with a gripper. A wrist-mounted RGB camera observes local interactions, and a global RGB camera at the front right of the workspace observes the scene and arm. The platform and representative task objects are shown in Appendix E (Figure 7). We fine-tune a task-specific $\pi_{0.5}$ policy using 50–100 teleoperated expert demonstrations per task (Physical Intelligence et al., 2025). The four tasks are: Task 1, place a yellow cube in a green bowl; Task 2, place a shovel and a badminton shuttlecock in a transparent storage box; Task 3, stack a red cube on a yellow cube; and Task 4, place a test tube in the upper drawer. Each method is evaluated in 20 trials per task, or 80 trials per method and 160 executions in total. A trial counts as successful only if the task is completed in full.

5.2 Base Models and Comparison Methods

OpenVLA-OFT and $\pi_{0.5}$ are the base policies in our LIBERO experiments. OpenVLA-OFT adapts OpenVLA to continuous regression, parallel decoding, and action chunking (Kim et al., 2025). Real-robot experiments use $\pi_{0.5}$ as the base policy; its flow-matching action expert is conditioned on robot state and action variables (Physical Intelligence et al., 2025; Black et al., 2025; Lipman et al., 2022). For both policies, we preserve the original action interfaces and trained weights and apply independent per-view cache and budget logic only where visual token reduction is supported. We compare against the base models and six visual token compression or VLA acceleration methods. SparseVLM uses text-guided sparsification (Zhang et al., 2025); DivPrune maximizes representation diversity (Alvar et al., 2025); FastV removes low-impact tokens using shallow-layer attention (Chen et al., 2024); VLA-Cache reuses temporally stable representations (Xu et al., 2025); VLA-Pruner combines semantic relevance with temporally smoothed action relevance (Liu et al., 2025); and SpecPrune-VLA uses action-aware signals to control action-level and network-level pruning (Wang et al., 2026). All LIBERO results use the common evaluation protocol described above.

Table 1: LIBERO success rates (%), normalized language-model FLOPs, and model-internal latency. FLOPs are normalized separately within each base-model group. Higher success and lower FLOPs and latency are better.

Method	Spatial	Object	Goal	Long	Avg. success	FLOPs	Latency
OpenVLA-OFT	92.80	98.60	96.20	90.60	94.55	100%	73.12 ms
FastV	89.60	93.60	96.00	90.60	92.45	57%	53.22 ms
SparseVLM	90.40	93.60	94.40	89.60	92.00	77%	57.19 ms
DivPrune	91.60	93.80	88.80	83.40	89.40	54%	54.47 ms
VLA-Cache	92.00	99.40	96.80	95.00	95.80	83%	69.85 ms
VLA-Pruner	90.60	99.40	97.80	92.20	95.00	42%	70.37 ms
SpecPrune-VLA	92.40	97.80	96.00	92.40	94.65	43%	58.54 ms
VADP-VLA	92.40	99.40	96.40	92.80	95.25	41%	48.89 ms
$\pi_{0.5}$	97.60	98.40	97.40	91.80	96.30	100%	37.38 ms
$\pi_{0.5}$+VADP-VLA	95.80	98.80	93.60	90.40	94.65	45%	25.43 ms

Table 2: Real-robot task success and VLA model-call latency. Each method is evaluated in 20 trials per task, or 80 trials across four tasks. Average success is an unweighted macro-average over tasks. Task 1–4 are defined in Section 5.1.

Method	Task 1	Task 2	Task 3	Task 4	Avg. success	Latency
$\pi_{0.5}$	20/20 (100%)	18/20 (90%)	16/20 (80%)	13/20 (65%)	83.75%	59.93 ms
VADP-VLA	20/20 (100%)	18/20 (90%)	15/20 (75%)	10/20 (50%)	78.75%	42.58 ms

5.3 Simulation Results

Table 1 compares task success, normalized language-model FLOPs, and model-internal latency. Relative to the unpruned OpenVLA-OFT baseline, VADP-VLA raises average success from 94.55% to 95.25%, reduces normalized language-model FLOPs to 41%, and lowers latency from 73.12 ms to 48.89 ms ($1.50\times$ speedup). At similar normalized FLOPs, VLA-Pruner achieves 95.00% success at 70.37 ms, compared with 95.25% at 48.89 ms for VADP-VLA. VLA-Cache reaches a higher average success rate of 95.80%, but uses 83% normalized FLOPs. With $\pi_{0.5}$ as the base model, VADP-VLA reduces normalized language-model FLOPs to 45% and latency from 37.38 ms to 25.43 ms ($1.47\times$ speedup), while average success decreases from 96.30% for the unpruned baseline to 94.65%. The largest suite-level success decrease occurs on LIBERO-Goal, from 97.40% to 93.60%; all suite-level results are reported in Table 1.

5.4 Real-Robot Results

Table 2 reports success counts and VLA inference latency for four tasks. Both $\pi_{0.5}$ and VADP-VLA succeed in $20/20$ trials on Task 1 and $18/20$ on Task 2. Success changes from $16/20$ to $15/20$ on Task 3 and from $13/20$ to $10/20$ on Task 4. Across 80 trials per method, the unpruned $\pi_{0.5}$ baseline succeeds $67/80$ times (83.75%), whereas VADP-VLA succeeds $63/80$ times (78.75%), a 5.00-percentage-point decrease in the point estimate. Under the same deployment environment and timing boundary, VADP-VLA reduces VLA model-call latency from 59.93 ms to 42.58 ms, a 17.35 ms or 28.95% reduction and approximately a $1.41\times$ speedup.

5.5 Ablation Studies

Independent per-view versus shared refresh. To assess independent refresh decisions, we compare shared refresh with independent per-view refresh. Table 3 shows success rates of 89% and 93%, respectively, a 4.00-percentage-point increase. Physical pruning rises from 70.87% to 73.13%, an increase of 2.26 points. Under the same pruning-ratio definition, independent refresh therefore has a higher success point estimate without increasing the average number of retained visual tokens. In the shared configuration, both views refresh at 10.91%; under independent refresh, the global and wrist rates are 7.97% and 9.08%, respectively. These results support a more favorable success–pruning trade-off for independent refresh in this comparison.

Table 3: Shared versus independent per-view refresh. Each configuration is evaluated in 100 trials; success is reported as both a count and a rate. Physical pruning is the fraction of visual tokens removed. Global- and wrist-view refresh rates describe how often the corresponding caches are updated.

Configuration	Success ↑	Physical pruning	Global refresh	Wrist refresh
Shared refresh	89/100 (89.00%)	70.87%	10.91%	10.91%
Independent per-view refresh	**93/100 (93.00%)**	**73.13%**	7.97%	9.08%

Table 4: Core-component ablation on LIBERO-Spatial. Latency is mean model-internal inference latency, and speedup is relative to the baseline in this ablation. Higher success and speedup are better; lower latency is better.

Configuration	Success ↑	Latency ↓	Speedup ↑
Baseline	92.80%	73.12 ms	1.00×
Baseline + Raw-QK pruning	90.80%	46.75 ms	1.56×
Baseline + Raw-QK + cache refresh	91.60%	48.44 ms	1.51×
Baseline + Raw-QK + wrist adaptive budget	92.60%	51.85 ms	1.41×
Full VADP-VLA	92.40%	48.89 ms	1.50×

Table 5: Task success for different values of λ. We select $\lambda = 0.5$ as the default fusion weight.

λ	Success
0.00	92/100
0.25	93/100
0.50	**95/100**
0.75	93/100
1.00	90/100

Core-component ablation. We compare component configurations on LIBERO-Spatial in Table 4. Static Raw-QK pruning reduces mean latency from 73.12 to 46.75 ms but lowers success from 92.80% to 90.80%. Adding historical caching and refresh raises success to 91.60% at an additional 1.69 ms. The wrist-view adaptive-budget configuration reaches 92.60% success at 51.85 ms. Full VADP-VLA attains 92.40% success and 48.89 ms mean latency: compared with the wrist-view adaptive-budget configuration, success is 0.20 percentage points lower and latency 2.96 ms shorter; compared with baseline, success is 0.40 points lower with a 1.50× speedup. In this evaluation, the full configuration offers a favorable success–latency trade-off; additional visualizations and parameter sensitivity results appear in Appendix D.

6 Conclusion

We presented VADP-VLA, a training-free framework for view-adaptive visual token pruning in multi-camera VLA inference. It preserves camera identity during relevance transfer, cache refresh, and budget allocation while decoupling cache invalidation from full-sequence computation. On LIBERO, VADP-VLA achieves 95.25% average success with normalized language-model FLOPs at 41% and a 1.50× model-internal speedup. With $\pi_{0.5}$, it achieves 94.65% LIBERO average success (versus 96.30% for the baseline), 45% normalized FLOPs, and a latency reduction from 37.38 ms to 25.43 ms. Across four real-robot tasks with 80 trials per method, VADP-VLA succeeds in 63/80 trials versus 67/80 for the unpruned $\pi_{0.5}$ baseline, while model-call latency falls from 59.93 ms to 42.58 ms, approximately a 1.41× speedup. Thus, the method reduces inference latency in the evaluated real-robot tasks, with a modest decrease in success.

AI use statement

The authors used generative AI tools to assist in identifying potentially relevant literature, discussing and preliminarily checking the research ideas proposed by the authors, and drafting and editing parts of the manuscript. The authors independently verified all references and technical claims. All AI-generated text, suggestions, and other outputs incorporated into this study underwent at least two rounds of human review. The authors take full responsibility for the final content of the paper.

Reproducibility statement

We value transparency in the research process and reproducibility of experimental results. To help other researchers understand and verify the method, the main paper and appendix specify visual

saliency estimation, per-view cache refresh, token budget allocation, and the inference algorithm. The state thresholds are $(\tau_1, \tau_2, \tau_3) = (0.98, 0.96, 0.94)$, and the main- and wrist-view refresh thresholds are $\gamma_m = 0.90$ and $\gamma_w = 0.94$, respectively. The first policy call retains all visual tokens; thereafter, either view retains 80% of its tokens when its cache is refreshed. For each base model, we report LIBERO success, language-model FLOPs normalized to the corresponding unpruned baseline, and measured inference latency. Real-robot results include the number of trials and successes for each task. Upon acceptance, we will make all source code developed for this work publicly available, including the implementation of VADP-VLA and the evaluation scripts. These protocols, parameter settings, and planned code release are intended to support independent verification and further study.

REFERENCES

Saeed Ranjbar Alvar, Gursimran Singh, Mohammad Akbari, and Yong Zhang. DivPrune: Diversity-based visual token pruning for large multimodal models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. arXiv:2503.02175.

Kevin Black et al. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems*, 2025. arXiv:2410.24164.

Anthony Brohan et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 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 *Proceedings of the European Conference on Computer Vision*, 2024. arXiv:2403.06764.

Ankit Goyal, Jie Xu, Yijie Guo, Valts Blukis, Yu-Wei Chao, and Dieter Fox. RVT: Robotic view transformer for 3d object manipulation. *arXiv preprint arXiv:2306.14896*, 2023.

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

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

Moo Jin Kim et al. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

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

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, 2023. arXiv:2306.03310.

Ziyan Liu, Yeqiu Chen, Hongyi Cai, Tao Lin, Shuo Yang, Zheng Liu, and Bo Zhao. Bridging the semantic-action gap in visual token pruning for efficient VLA inference. *arXiv preprint arXiv:2511.16449*, 2025.

Shilin Ma, Chubin Zhang, Changyuan Wang, Yuji Wang, Yue Wu, Zixuan Wang, Jingqi Tian, Zheng Zhu, and Yansong Tang. SAFE-Pruner: Semantic attention-guided future-aware token pruning for efficient vision-language-action manipulation. In *Proceedings of the European Conference on Computer Vision*, 2026. arXiv:2605.29662.

Octo Model Team et al. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, 2024. arXiv:2405.12213.

Yuhao Pan, Haosong Peng, Zhengshen Zhang, Zhengyang Yan, Yalun Dai, Fushuo Huo, Chujie Wang, Tianyu Qi, Xiucheng Wang, Nan Cheng, and Wenchao Xu. World-to-wrist: Task-conditioned future wrist modeling for fine-grained robot manipulation. *arXiv preprint arXiv:2608.05369*, 2026.

Physical Intelligence, Kevin Black, Noah Brown, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Yuzhang Shang, Mu Cai, Bingxin Xu, Yong Jae Lee, and Yan Yan. LLaVA-PruMerge: Adaptive token reduction for efficient large multimodal models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025. arXiv:2403.15388.

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *Proceedings of the Conference on Robot Learning*, 2022. arXiv:2209.05451.

Hanzhen Wang, Jiaming Xu, Yushun Xiang, Jiayi Pan, Yongkang Zhou, Yong-Lu Li, and Guohao Dai. SpecPrune-VLA: Accelerating vision-language-action models via action-aware self-speculative pruning. In *Proceedings of the International Conference on Machine Learning*, 2026. arXiv:2509.05614.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. VLA-Cache: Efficient vision-language-action manipulation via adaptive token caching. In *Advances in Neural Information Processing Systems*, 2025. arXiv:2502.02175.

Jiarui Yang, Yehao Lu, Yuning Su, Yufeng Xie, Yu Zhong, Haiyu Lan, Tianjing Hao, Kaixiang Lu, Peiwen Lin, Chuang Wang, Enyu Li, and Junwei Liang. Multi-view unified camera fields: Geometry-shaped action-facing representations for RGB-only multi-camera VLA policies. *arXiv preprint arXiv:2608.01826*, 2026.

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*, 2025. arXiv:2412.04467.

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 *Advances in Neural Information Processing Systems*, 2024. arXiv:2411.02359.

Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, and Shanghang Zhang. SparseVLM: Visual token sparsification for efficient vision-language model inference. In *Proceedings of the International Conference on Machine Learning*, 2025. arXiv:2410.04417.

A Detailed Inference Algorithm

Algorithm 1 summarizes a complete VADP-VLA policy call. Let $\mathbf{H}_t^{(\ell)}$ be the hidden states after the first ℓ decoder modules. For each view, the algorithm uses the input normalization and query–key projections of module $\ell+1$ to compute current visual saliency. It then matches features to that view's latest keyframe and computes agreement between current and historical saliency. If agreement falls below the view's refresh threshold γ_v, its keyframe is refreshed. The complete projected visual tokens and saliency of the current frame become the new keyframe, while token selection uses the maximum retention ratio $q_{\mathrm{ref}} = 0.8$. Otherwise, the existing keyframe is retained and the retention ratio is selected from the four budget states according to current saliency agreement. Finally, TopK selection is performed within each view. Selected visual hidden states and all non-visual hidden states are combined in original sequence order, and the compressed sequence is passed to decoder module $\ell + 1$ and subsequent modules.

B Complexity Analysis

This section analyzes the effect of VADP-VLA on language-model backbone computation. Let the language model have L decoder modules, each with hidden dimension d and feed-forward intermediate dimension d_{ff}. At policy call t, the full multimodal sequence contains all visual and non-visual tokens and has length

$$N = N_{\mathrm{nv}} + N_{\mathrm{vis}} = N_{\mathrm{nv}} + N_m + N_w, \tag{8}$$

Here, $N_{\mathrm{nv}} = |\Omega_{\mathrm{nv}}|$ is the number of non-visual tokens, including text, proprioception, and action queries, while N_m and N_w are the token counts for the main and wrist views. In our OpenVLA-OFT configuration, $N_m = N_w = 256$, so $N_{\mathrm{vis}} = 512$.

For an input sequence of length n, the computation of a standard Transformer decoder module can be approximated as

$$\mathcal{C}(n) = 4nd^2 + 2n^2d + 3ndd_{\mathrm{ff}}. \tag{9}$$

The $4nd^2$ term accounts for query, key, value, and output projections; $2n^2d$ accounts for attention-score computation and attention-weighted aggregation; and $3ndd_{\mathrm{ff}}$ accounts for the three linear projections in a gated feed-forward network. This expression characterizes how sequence length affects backbone computation and omits lower-order terms such as layer normalization, biases, and elementwise operations.

For the unpruned OpenVLA-OFT baseline, all L decoder modules process the full length-N sequence, so total language-model backbone FLOPs are

$$F_{\mathrm{base}}^{\mathrm{LM}} = L\,\mathcal{C}(N). \tag{10}$$

VADP-VLA processes the full sequence through the first $\ell = 3$ decoder modules. Unselected visual hidden states are physically removed before subsequent modules. Under the per-view budgets in Section 4.4, the retained visual token count is

$$k_t^v = \begin{cases} N_v, & t = 1, \\ \mathrm{round}(q_t^v N_v), & t > 1, \end{cases} \qquad v \in \{m, w\}, \tag{11}$$

where q_t^v is the retention ratio for view v at call t. The compressed sequence entering later decoder modules therefore has length

$$\widehat{N}_t = N_{\mathrm{nv}} + k_t^m + k_t^w. \tag{12}$$

Excluding auxiliary operations such as token scoring, historical matching, and budget control, language-model backbone FLOPs for VADP-VLA at call t are

$$F_{\mathrm{VADP},t}^{\mathrm{LM}} = \ell\,\mathcal{C}(N) + (L - \ell)\mathcal{C}(\widehat{N}_t). \tag{13}$$

The theoretical FLOPs reduction relative to the unpruned baseline is

$$\Delta_t^{\mathrm{LM}} = 1 - \frac{F_{\mathrm{VADP},t}^{\mathrm{LM}}}{F_{\mathrm{base}}^{\mathrm{LM}}} = \frac{L - \ell}{L}\left[1 - \frac{\mathcal{C}(\widehat{N}_t)}{\mathcal{C}(N)}\right]. \tag{14}$$

The corresponding ideal backbone speedup is

$$S_t^{\mathrm{ideal}} = \frac{F_{\mathrm{base}}^{\mathrm{LM}}}{F_{\mathrm{VADP},t}^{\mathrm{LM}}}. \tag{15}$$

The physical visual token pruning ratio differs from the compressed length of the complete multimodal sequence. Counting visual tokens only, the physical pruning ratio at call t is

$$r_t^{\mathrm{vis}} = 1 - \frac{k_t^m + k_t^w}{N_m + N_w}. \tag{16}$$

If $N_m = N_w$ and rounding error is ignored, then

$$r_t^{\mathrm{vis}} \approx 1 - \frac{q_t^m + q_t^w}{2}. \tag{17}$$

Table 6: Comparison of query token sources. Both configurations use the same checkpoint, visual keys, pruning mechanism, cache policy, token budget controller, and evaluation protocol; only the query-token source differs.

Query source	Success (%) ↑	Model-internal latency (ms) ↓
Text tokens as queries	95.25 (1905/2000)	48.89
Action query tokens as queries	94.75 (1895/2000)	66.46

Thus, the two views may use different token budgets and contribute separately to the overall visual pruning ratio.

Beyond the language-model backbone, VADP-VLA performs task-relevance scoring, within-view feature matching, and Top-K selection. Counted separately, their additional computation is

$$F_{\mathrm{aux},t} = F_{\mathrm{score},t} + F_{\mathrm{match},t} + F_{\mathrm{select},t}, \tag{18}$$

The asymptotic complexities of saliency probing, within-view matching, and token selection, respectively, can be written as

$$F_{\mathrm{score},t} = \mathcal{O}(N_h L_Q N_{\mathrm{vis}} d_h)\,, \qquad F_{\mathrm{match},t} = \mathcal{O}\big((N_m^2 + N_w^2)d\big)\,, \qquad F_{\mathrm{select},t} = \mathcal{O}(N_{\mathrm{vis}}). \tag{19}$$

Here, N_h is the number of attention heads, L_Q is the number of text queries used for relevance computation, and d_h is the dimension per head. These auxiliary operations run only during token selection, and their actual overhead depends on implementation and hardware kernels.

C Experimental Details

C.1 Query Token Selection

The raw query–key relevance score in Section 4.2 requires a query sequence that reflects the task condition without unnecessary computation. We compare two sources: (i) *non-visual, non-action text tokens* associated with the language instruction, and (ii) *action query tokens* used by the continuous action-regression branch. Visual keys, the relevance estimator, temporal cache, refresh thresholds, token budget controller, and evaluation protocol are held fixed; only the source of queries for visual relevance changes.

Table 6 reports aggregate results under the same LIBERO evaluation protocol. Text queries achieve 95.25% success $(1905/2000)$ versus 94.75% $(1895/2000)$ for action queries, while model-internal latency falls from 66.46 ms to 48.89 ms. The lower latency is consistent with the shorter query sequence and avoids involving the action-regression branch in visual token ranking. We therefore use non-visual, non-action text tokens as queries in all main experiments.

C.2 View-Specific Token Budget Configuration

In the main text, q_t^v is the visual token retention ratio for view v at policy call t; the corresponding pruning ratio is $1 - q_t^v$. We use $q_{\mathrm{I}}^v, \ldots, q_{\mathrm{IV}}^v$ for candidate retention ratios in states I–IV. In the implementation, each view has its own budget mapping: the wrist view has four levels and the global view has two. We denote the global and wrist views by $v = m$ and $v = w$, respectively.

State and refresh thresholds. We use

$$(\tau_1, \tau_2, \tau_3) = (0.98,\ 0.96,\ 0.94)$$

to divide saliency agreement ρ_t^v: state I has $\rho_t^v \geq 0.98$, state II has $0.96 \leq \rho_t^v < 0.98$, state III has $0.94 \leq \rho_t^v < 0.96$, and state IV has $\rho_t^v < 0.94$. The view-specific refresh thresholds are $\gamma_m = 0.90$ and $\gamma_w = 0.94$. When $\rho_t^v < \gamma_v$, view v first refreshes its keyframe and sets its retention ratio to $q_t^v = 0.8$. The view-specific state budget is used only if no refresh occurs. Refresh thus takes precedence over the state budget and does not restore the full visual sequence.

Wrist-view budget. In the absence of refresh, the wrist view has four candidate retention ratios:

$$(q_{\mathrm{I}}^{w}, q_{\mathrm{II}}^{w}, q_{\mathrm{III}}^{w}, q_{\mathrm{IV}}^{w}) = (0.2,\ 0.4,\ 0.6,\ 0.8). \tag{20}$$

The corresponding pruning ratios are

$$(1 - q_{\mathrm{I}}^{w},\ 1 - q_{\mathrm{II}}^{w},\ 1 - q_{\mathrm{III}}^{w},\ 1 - q_{\mathrm{IV}}^{w}) = (0.8,\ 0.6,\ 0.4,\ 0.2).$$

With high saliency agreement, the wrist view uses a lower retention ratio. As agreement falls, it retains progressively more tokens to cover visual changes caused by arm motion, approach to a target, object contact, and local occlusion. If the view refreshes, the state budget is bypassed and the retention ratio is set directly to 0.8.

Global-view budget. In the absence of refresh, the global view maps four states to two retention levels:

$$(q_{\mathrm{I}}^{m}, q_{\mathrm{II}}^{m}, q_{\mathrm{III}}^{m}, q_{\mathrm{IV}}^{m}) = (0.1,\ 0.1,\ 0.8,\ 0.8). \tag{21}$$

States I and II thus retain approximately 10% of visual tokens, while states III and IV retain approximately 80%, corresponding to pruning ratios of about 90% and 20%. The global view primarily provides workspace and global position information. Low retention reduces repeated computation when agreement is high; when agreement falls, higher retention preserves visual evidence about changes in position and layout. The wrist view is more affected by arm motion and local interaction, motivating finer-grained control with four budget levels. A refreshed global view likewise uses a retention ratio of 0.8 directly.

State numbering has the same meaning for both views: state I has the highest saliency agreement and lowest inferred update requirement, while state IV has the lowest agreement and highest inferred update requirement. These states do not correspond to predefined semantic task phases such as grasping, moving, or placing.

Actual retained token counts. In our OpenVLA-OFT configuration, each view contains $N_v = 256$ visual tokens. The four wrist-view budgets correspond to retained counts of

$$\mathrm{round}(0.2 \times 256) = 51, \qquad \mathrm{round}(0.4 \times 256) = 102,$$

$$\mathrm{round}(0.6 \times 256) = 154, \qquad \mathrm{round}(0.8 \times 256) = 205.$$

The two global-view budgets correspond to

$$\mathrm{round}(0.1 \times 256) = 26, \qquad \mathrm{round}(0.8 \times 256) = 205.$$

After rounding, the actual pruning ratios for the two global-view budgets are approximately 89.8% and 19.9%; for the four wrist-view budgets, they are approximately 80.1%, 60.2%, 39.8%, and 19.9%. The main text refers to these as approximately 90% and 20% for the global view and 80%, 60%, 40%, and 20% for the wrist view.

The first policy call initializes both keyframe caches and retains all 256 visual tokens in each view. From the second call onward, each view independently decides whether to refresh. On refresh, the cache is updated with the complete current-view information, but only $\mathrm{round}(0.8N_v)$ visual tokens enter later decoder modules. Otherwise, the state budget is selected from that view's saliency agreement. Thus, the two views may use different retention ratios and token counts in the same policy call.

C.3 Comparison Models and Baselines

To assess the success–efficiency trade-off of VADP-VLA, we compare it with the unpruned OpenVLA-OFT baseline, general visual token compression methods, and efficient inference methods for VLA models. These methods differ in where they intervene in computation, which decision signals they use, whether they use information across policy calls, and whether they explicitly distinguish camera views. Methods in the main table are evaluated on the same LIBERO suites and task partitions. Language-model FLOPs are normalized to 100% against the corresponding unpruned baseline within each base-model group; normalized FLOPs for the OpenVLA-OFT and $\pi_{0.5}$ groups do not directly compare their absolute compute.

Unpruned OpenVLA-OFT baseline. OpenVLA-OFT is our primary base policy and compute baseline. It retains all visual tokens from both cameras and processes the complete visual token sequence in every language-model module. Apart from visual token handling, VADP-VLA does not change OpenVLA-OFT's model weights, visual encoder, action output head, action-chunk length, or continuous action-regression interface. The unpruned OpenVLA-OFT baseline therefore measures the changes in success and computation associated with visual token compression.

FastV. FastV is a general vision-language model acceleration method that estimates visual token importance from attention signals in shallow Transformer modules and removes low-impact tokens from later computation (Chen et al., 2024). It can be integrated with little additional machinery and does not require retraining the original model. We use FastV as a general shallow-saliency pruning baseline. Unlike VADP-VLA, it makes its main decisions over the concatenated visual token sequence and does not explicitly maintain separate historical saliency states or agreement-based budgets for the global and wrist views.

SparseVLM. SparseVLM sparsifies tokens according to instruction-guided visual relevance, prioritizing visual information relevant to the current text condition (Zhang et al., 2025). It represents instruction-conditioned visual token selection. Its main decision signal comes from the current image–text request rather than camera-specific temporal state across policy calls. SparseVLM therefore helps compare single-call text-conditioned saliency selection with the per-view temporal selection in VADP-VLA.

DivPrune. DivPrune reduces visual token counts by maximizing diversity among retained visual features (Alvar et al., 2025). Compared with pruning based solely on relevance rankings, it tends to retain features covering different visual content and thereby reduces representational redundancy in selection. We use DivPrune as a diversity-driven compression baseline. It focuses on coverage of a single set of visual features and lacks the within-view historical matching, independent refresh gate, and view-adaptive budget in VADP-VLA.

VLA-Pruner. VLA-Pruner combines visual semantic relevance with action relevance and uses temporal information to smooth pruning decisions during VLA inference (Liu et al., 2025). Its decision signal is closer to robot-control objectives than those of FastV and SparseVLM, making it an action-aware visual token pruning method for VLA models. In our comparison, VLA-Pruner probes the effect of action relevance and temporal smoothing on inference efficiency. Its main focus is semantic and action relevance, whereas VADP-VLA additionally maintains temporal state per camera and independently controls each camera's cache refresh and token budget.

SpecPrune-VLA. SpecPrune-VLA uses action-aware self-speculation to control computation at both the action and network levels (Wang et al., 2026). It is therefore a VLA acceleration framework that addresses both action prediction and the network execution path, rather than a standalone visual token ranking method. In our comparison, it represents hierarchical, action-aware VLA acceleration. By contrast, VADP-VLA keeps language-model depth and the action output interface fixed and reduces sequence computation per call mainly by physically removing visual hidden states before later modules.

VLA-Cache. VLA-Cache follows a different acceleration paradigm. It caches and reuses temporally stable visual representations across consecutive policy calls to reduce repeated feature computation (Xu et al., 2025). Its core operation is cross-call information caching and compute reuse, rather than removing visual tokens according to importance scores. Thus, VLA-Cache should not be classified as visual token pruning, and its normalized FLOPs in the main table cannot be interpreted directly as a visual token pruning ratio.

We include VLA-Cache in the main table because it shares the overall goal of reducing computation in repeated multi-view VLA inference. The two methods take different routes: VLA-Cache mainly reduces repeated representation computation across policy calls, while VADP-VLA uses token selection and physical sequence compression to reduce computation in later language-model modules within each call. Cache storage, feature matching, invalidation, and data movement also incur overhead; therefore, model-internal latency and FLOPs for a caching method must be measured under the common experimental protocol rather than inferred from visual token counts alone.

Table 7: Mechanism categories for the comparison methods. Per-view control indicates whether a method explicitly maintains separate temporal state, refresh decisions, or token budgets for different cameras.

Method	Intervention	Decision signal	Cross-call information	Per-view control
OpenVLA-OFT	-	No pruning	No	No
FastV	Visual tokens	Shallow attention saliency	No	No
SparseVLM	Visual tokens	Text-conditioned relevance	No	No
DivPrune	Visual tokens	Feature diversity	No	No
VLA-Pruner	Visual tokens	Semantic and action relevance	Limited temporal information	Not explicit
SpecPrune-VLA	Action and network execution	Action-aware self-speculation	Setting-dependent	Not explicit
VLA-Cache	Cross-call representation reuse	Temporal stability	Yes	Outside pruning paradigm
VADP-VLA	Visual tokens	Task relevance and view agreement	Per-view cache	Yes

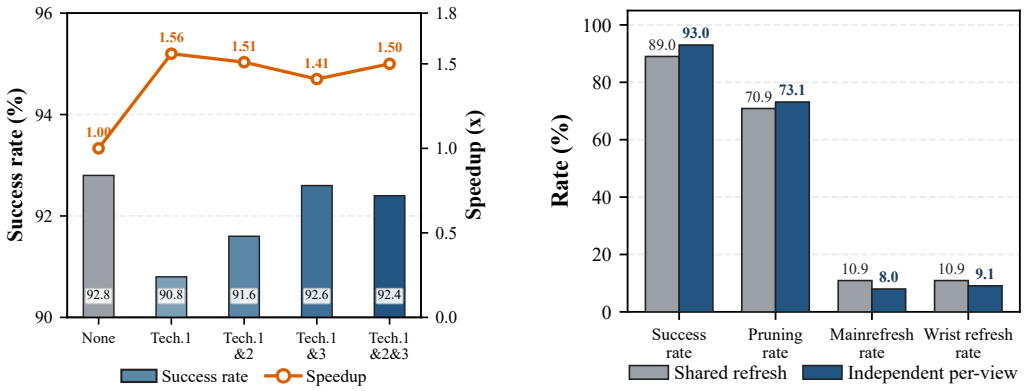
VADP-VLA. VADP-VLA is a training-free framework for view-adaptive visual token pruning. After processing the full sequence through the first $\ell = 3$ language-model modules, it estimates task-conditioned visual relevance from shallow hidden states. The main and wrist views then maintain separate within-view historical saliency caches and independently decide whether to refresh according to their temporal agreement. Each view maps to one of four visual update states and chooses a token budget from its retention-ratio set. Unselected visual hidden states are physically removed before later language-model modules, while all non-visual tokens and the action output interface remain unchanged.

The main distinctions are as follows. FastV uses shallow saliency, SparseVLM uses text-conditioned relevance, DivPrune uses visual feature diversity, VLA-Pruner combines semantic and action relevance, SpecPrune-VLA jointly controls action and network execution, and VLA-Cache reduces repeated computation through cross-call reuse. VADP-VLA integrates task-conditioned relevance, within-view historical saliency, per-view refresh, and per-view dynamic budgets in one visual token pruning procedure.

Table 7 illustrates mechanistic differences only; FLOPs from different method categories cannot be interpreted as the same visual token pruning ratio. Final success, normalized language-model FLOPs, and model-internal latency follow the unified evaluation results in Table 1.

D Additional Ablation Results

D.1 Component and Refresh Comparisons



(a) Ablation on different techniques.

(b) View control comparison.

Figure 4: (a) Core-component ablation on LIBERO-Spatial. Bars show task success, and the line shows inference speedup relative to the baseline in this ablation. Exact values are given in Table 4. (b) Comparison of shared and independent per-view refresh. Independent refresh raises success from 89% to 93% and physical pruning from 70.87% to 73.13%, while reducing global- and wrist-view refresh rates from 10.91% to 7.97% and 9.08%, respectively. Per-view control thus reduces refresh frequency while yielding a higher pruning ratio and better task performance in this comparison.

D.2 Parameter Sensitivity and Token Retention

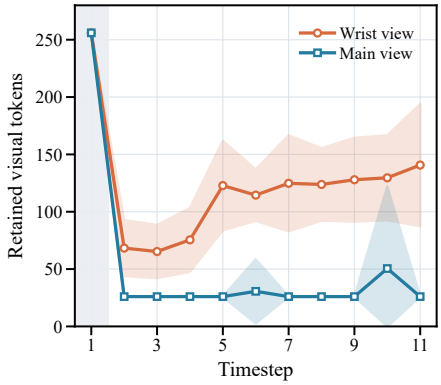


Figure 5: **Visual token pruning in a single task.** Retained token counts for the global and wrist views during task execution. Solid lines show the mean retained count at each time step, and light bands show the mean ± one standard deviation.

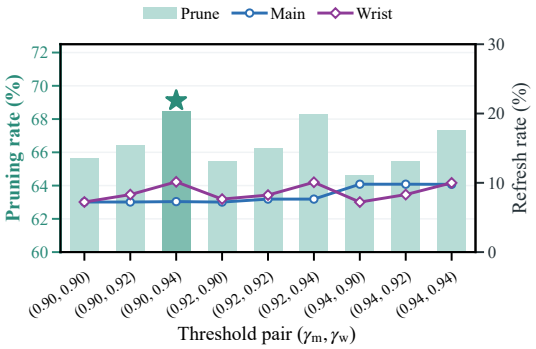


Figure 6: **Ablation of refresh thresholds γ_m and γ_w.** The vertical axis shows the pruning ratio and global- and wrist-view refresh rates. Among the evaluated combinations, $\gamma_m = 0.90$ and $\gamma_w = 0.94$ give the highest physical pruning ratio among settings with the same $95/100$ success rate.

Sensitivity to key parameters. We examine how cache-refresh thresholds and the historical-saliency fusion weight affect task success and token compression. For the refresh-threshold sweep, we select each of γ_m and γ_w from $\{0.90, 0.92, 0.94\}$, evaluating nine combinations. Figure 6 shows that every configuration succeeds in 95 of 100 trials, but retained token counts and cache-refresh frequencies differ. With $\gamma_m = 0.90$ fixed, increasing γ_w from 0.90 to 0.94 raises wrist-view refresh from 7.23% to 10.14%, reduces the mean retained count from 133.43 to 118.72, and raises overall physical pruning from 65.61% to 68.46%. By contrast, with $\gamma_w = 0.94$ fixed, increasing γ_m raises global-view refresh frequency and lowers overall pruning. We therefore use $\gamma_m = 0.90$ and $\gamma_w = 0.94$ by default; this pair preserves the same success in the sweep and gives the highest physical pruning ratio. For the fusion weight, $\lambda = 0$ and $\lambda = 1$ use only current and only transferred historical saliency, respectively, and yield $92/100$ and $90/100$ successes in Table 5. Both $\lambda = 0.25$ and $\lambda = 0.75$ yield $93/100$, while $\lambda = 0.5$ reaches $95/100$. These results support combining current and historical information, so we use $\lambda = 0.5$ by default.

E Real-Robot Task Examples

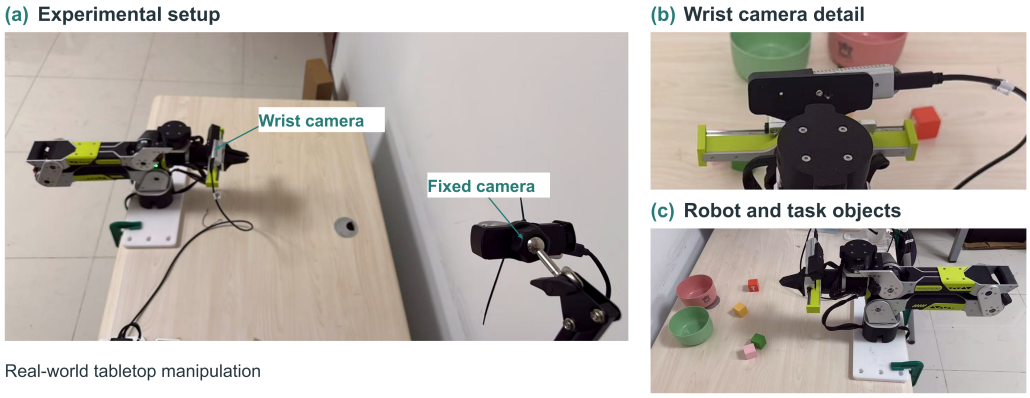
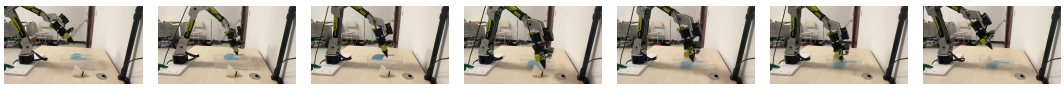


Figure 7: Real-robot experimental platform. A wrist-mounted RGB camera and a fixed global RGB camera observe the Seeed Studio Arm B601-DM robot arm. Panels show the overall workspace, wrist camera, and representative task objects.

Figure 8 shows representative observation frames from the four real-robot tasks and their execution. Each row corresponds to one task instruction, with frames ordered left to right. The tasks are placing a yellow cube in a green bowl, placing a shovel and a badminton shuttlecock in a transparent box, stacking a red cube on a yellow cube, and placing a test tube in the upper drawer. These examples cover pick-and-place, multi-object organization, block stacking, and drawer interaction, and depict different objects, containers, and spatial relations in real scenes.



(a) Task 1: Put the yellow cube into the green bowl.



(b) Task 2: Put the shovel and the badminton shuttlecock into the transparent box.



(c) Task 3: Stack the red cube onto the yellow cube.



(d) Task 4: Place the test tube in the top drawer.

Figure 8: Sequential key frames for four robotic manipulation tasks. Each row illustrates the full execution process of one task.

Algorithm 1 Per-view visual token pruning in VADP-VLA

Input: Policy-call index t; hidden states $\mathbf{H}_t^{(\ell)}$ after the first ℓ decoder modules; complete projected visual tokens $\{\mathbf{E}_t^v\}_{v\in\{m,w\}}$; per-view keyframe index k_v and cache $\mathcal{C}_{k_v}^v = (\mathbf{E}_{k_v}^v, \mathbf{s}_{k_v}^v)$ (empty on the first call)
Input: Fusion weight λ; refresh thresholds γ_m, γ_w; state thresholds $\tau_1 > \tau_2 > \tau_3$
Input: Wrist-view budgets $(q_{\mathrm{I}}^w, q_{\mathrm{II}}^w, q_{\mathrm{III}}^w, q_{\mathrm{IV}}^w) = (0.2, 0.4, 0.6, 0.8)$
Input: Main-view budgets $(q_{\mathrm{I}}^m, q_{\mathrm{II}}^m, q_{\mathrm{III}}^m, q_{\mathrm{IV}}^m) = (0.1, 0.1, 0.8, 0.8)$; refresh retention ratio $q_{\mathrm{ref}} = 0.8$
Output: Compressed hidden states $\widehat{\mathbf{H}}_t^{(\ell)}$; updated keyframe indices and caches

for each view $v \in \{m, w\}$ **do**
  Compute current saliency $\mathbf{s}_t^v$ using Eq. equation 2
  if $t = 1$ **then**
    $q_t^v \leftarrow 1$
    $k_t^v \leftarrow N_v$
    $\mathcal{K}_t^v \leftarrow \{1, \ldots, N_v\}$
    $k_v \leftarrow t$
    $\mathcal{C}_{k_v}^v \leftarrow (\mathbf{E}_t^v, \mathbf{s}_t^v)$
  else
    for $i = 1, \ldots, N_v$ **do**
      $j_{t,i}^{v*} \leftarrow \arg\max_{1\le j\le N_v} \cos(\mathbf{e}_{t,i}^v, \mathbf{e}_{k_v,j}^v)$
      $\widetilde{s}_{t,i}^v \leftarrow s_{k_v, j_{t,i}^{v*}}^v$
    end for
    $\rho_t^v \leftarrow \cos(\mathbf{s}_t^v, \widetilde{\mathbf{s}}_t^v)$
    if $\rho_t^v < \gamma_v$ **then**
      ▷ Refresh takes precedence over the state budget
      $g_t^v \leftarrow 1$
      $q_t^v \leftarrow q_{\mathrm{ref}} = 0.8$
      $\mathbf{s}_t^{v,\mathrm{sel}} \leftarrow \mathbf{s}_t^v$
      $k_v \leftarrow t$
      $\mathcal{C}_{k_v}^v \leftarrow (\mathbf{E}_t^v, \mathbf{s}_t^v)$
    else
      $g_t^v \leftarrow 0$
      Retain keyframe index k_v and cache $\mathcal{C}_{k_v}^v$
      $\mathbf{s}_t^{v,\mathrm{sel}} \leftarrow \mathcal{N}((1-\lambda)\mathbf{s}_t^v + \lambda\widetilde{\mathbf{s}}_t^v)$
      if $v = w$ **then**
        if $\rho_t^v \ge \tau_1$ **then**
          $q_t^v \leftarrow q_{\mathrm{I}}^w = 0.2$
        else if $\rho_t^v \ge \tau_2$ **then**
          $q_t^v \leftarrow q_{\mathrm{II}}^w = 0.4$
        else if $\rho_t^v \ge \tau_3$ **then**
          $q_t^v \leftarrow q_{\mathrm{III}}^w = 0.6$
        else
          $q_t^v \leftarrow q_{\mathrm{IV}}^w = 0.8$
        end if
      else
        if $\rho_t^v \ge \tau_2$ **then**
          $q_t^v \leftarrow 0.1$
          ▷ Main-view states I and II
        else
          $q_t^v \leftarrow 0.8$
          ▷ Main-view states III and IV
        end if
      end if
    end if
    $k_t^v \leftarrow \mathrm{round}(q_t^v N_v)$
    $\mathcal{K}_t^v \leftarrow \mathrm{TopK}\left(\mathbf{s}_t^{v,\mathrm{sel}}, k_t^v\right)$
  end if
end for
Map $\mathcal{K}_t^m$ and $\mathcal{K}_t^w$ to visual-token positions in the original multimodal sequence
Retain selected visual hidden states and all non-visual hidden states from $\mathbf{H}_t^{(\ell)}$ in original sequence order to obtain $\widehat{\mathbf{H}}_t^{(\ell)}$
Update position indices and the attention mask consistently; pass $\widehat{\mathbf{H}}_t^{(\ell)}$ to decoder module $\ell + 1$ and subsequent modules