MEMCACHE: MEMORY-AUGMENTED ADAPTIVE TOKEN CACHING FOR EFFICIENT VISION-LANGUAGE-ACTION INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-Language-Action (VLA) policies facilitate generalized robotic manipulation by mapping visual scenes and task instructions to low-level robotic actions. Nevertheless, their heavy computations fail to meet the stringent real-time control constraints and limited computational budgets of physical robots. As an efficient inference paradigm, VLA caching alleviates computational burdens by exploiting temporal redundancy across consecutive robotic observations and reusing static token states. However, existing methods exhibit several limitations: their caching decisions generally depend on hand-crafted proxy signals often misaligned with control policy demands, and they rely only on adjacent frames without exploiting longer-term temporal context. To this end, we present MemCache, a novel learnable memory-augmented caching framework for high-efficiency and reliable VLA inference. Specifically, we leverage learnable query banks to generate token-level importance scores. This eliminates dependence on hand-crafted proxy supervision and externally defined signals, enabling the token-selection strategy to be fully optimized with the control objective. Additionally, we integrate a recurrent visual memory state to capture long-range temporal dependencies in manipulation behaviors, providing history-aware corrections for caching decisions. We further develop a linear-warmup training schedule to coordinate the above two components and facilitate joint optimization. Equipped with these merits, MemCache delivers reliable and efficient VLA inference for robotic manipulation tasks, e.g., $\uparrow \mathbf{0.7}\% \sim \uparrow \mathbf{1.2}\%$ higher task performance with $\uparrow 1.07\times \sim \uparrow \mathbf{1.45}\times$ speedups over VLA caching baselines on the LIBERO benchmark.

1 INTRODUCTION

Vision-language-action (VLA) policies map visual scenes and task instructions to low-level robotic actions, enabling unified models with broad generalization across objects, manipulation tasks, and robot platforms. For example, RT-2 transfers web-scale vision-language knowledge to robotics via action tokenization (Zitkovich et al., 2023), while large-scale robotic learning efforts (O'Neill et al., 2024; Octo Model Team et al., 2024) further verify that heterogeneous robot datasets produce strong generalist VLA policies. Recent works diversify VLA designs, covering autoregressive models for discrete action token generation (Kim et al., 2024) and flow-matching-based frameworks for continuous action prediction (Black et al., 2024). However, VLA inference incurs substantial computational costs (Li et al., 2024a; Liu et al., 2025). The resulting latency conflicts with the stringent real-time control-frequency requirements for physical robots (Xu et al., 2025; Wei et al., 2026).

To solve this issue, various methods have been proposed to reduce computational costs via different avenues for VLA inference, such as architectural optimizations (Liu et al., 2024b; Wen et al., 2025), quantization (Park et al., 2024; Wang et al., 2025b), and action-space optimizations (Pertsch et al., 2025; Kim et al., 2025). Among them, VLA caching has emerged as a promising solution, as it alleviates the inherent temporal redundancy in consecutive robotic observations (Xu et al., 2025; Wei et al., 2026). This paradigm computes key-value (KV) states only for dynamic and task-critical tokens, while reusing cached states for static and less informative tokens (see Fig. 1).

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.

Nevertheless, existing VLA caching methods still suffer from notable limitations. First, they typically depend on hand-crafted or externally defined proxy signals to identify critical tokens for computation, such as the text-to-vision attention mechanism adopted by VLA-Cache (Xu et al., 2025) and RAFT-small optical flow in LAC (Wei et al., 2026). Such proxy-based supervision often fails to align precisely with the true requirements of downstream control policies. Furthermore, existing methods make caching decisions solely based on adjacent observations (Xu et al., 2025; Wei et al., 2026), lacking a recurrent state to accumulate longer-term interaction context.

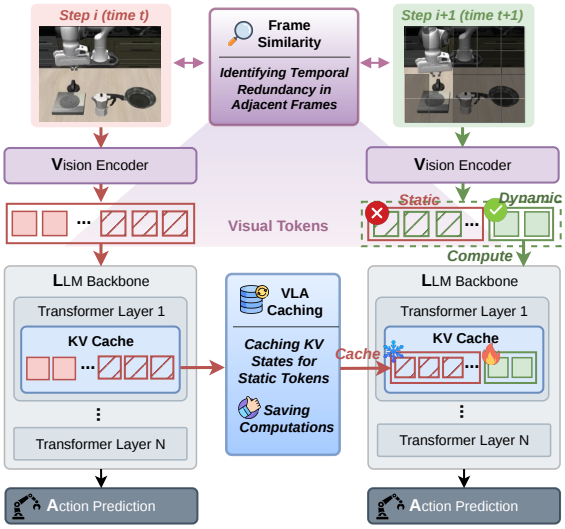

Figure 1: VLA caching exploits inherent temporal similarity across successive robotic steps by computing KV states only for important and dynamic tokens.

To tackle these limitations, we propose a novel learnable memory-augmented caching framework, termed **MemCache**. We first adopt a bank of learnable queries to perform cross-attention over projected visual tokens and produce query-to-visual importance representations. This design allows the token selection criterion to be optimized end-to-end with the control objective, removing the reliance on hand-crafted proxy supervision and externally defined signals. Additionally, we incorporate a recurrent visual memory state to accumulate temporal dependencies across successive manipulation steps. This history-aware context supplies corrective feedback to the underlying query-based caching mechanism, yielding memory-augmented caching decisions that jointly leverage instantaneous observations and long-range historical context for more reliable token caching. Finally, we develop a linear-warmup schedule coupled with a novel memory-consistency objective to orchestrate the aforementioned two core components. This mechanism progressively incorporates memory-conditioned corrections into query-based caching decisions and facilitate end-to-end joint optimization. We summarize our contributions as:

- We present MemCache, a memory-augmented adaptive caching framework for closed-loop robotic manipulation. It resolves key drawbacks of existing VLA caching methods: dependence on predefined or externally derived proxy signals as well as insufficient modeling of long-term temporal context.
- We first adopt learnable-query-based caching decisions to enable end-to-end optimization without hand-crafted proxies. Then, a recurrent memory state is developed to refine caching decisions via historical corrections. A linear-warmup training schedule paired with a novel memory consistency objective is further proposed to facilitate training.
- Benefiting from these innovative designs, MemCache achieves more reliable token caching for sequential robotic-manipulation tasks. Specifically, MemCache offers $\uparrow \mathbf{0.7}\% \sim \uparrow \mathbf{1.2}\%$ higher task performance alongside $\uparrow 1.07\times \sim \uparrow \mathbf{1.45}\times$ speedups over VLA caching baselines for OpenVLA and OpenVLA-OFT models under the LIBERO benchmark.

2 Related Work

Vision-Language-Action Models. VLA models extend multimodal foundation models from perception and language understanding (Alayrac et al., 2022; Li et al., 2023; Dai et al., 2023; Bai et al., 2023; Deng et al., 2025; Touvron et al., 2023) to closed-loop control (Zitkovich et al., 2023; Black et al., 2024; O'Neill et al., 2024). Early work such as PaLM-E injects embodied observations into a language model (Driess et al., 2023). Furthermore, robot-oriented adaptation of pretrained vision-language representations has demonstrated strong imitation-learning performance (Li et al., 2024b). Recent models diverge in representation and generation of control outputs. For example, autoregressive VLA discretizes actions into language-like tokens, directly reusing pre-trained language-model machinery at the cost of sequential decoding overhead (Pertsch et al., 2025; Zitkovich et al., 2023;

Kim et al., 2024). Other approaches adopt continuous action heads to model multimodal trajectories and enhance control smoothness, though they incur iterative computational burdens (Liu et al., 2025; Black et al., 2024; Li et al., 2024a). Across both model families, scaling the backbone and visual context boosts policy generality yet increases inference latency.

Acceleration of VLA Models. VLA efficiency has been pursued via many directions. Architectural optimizations streamline model backbones via replacement (Liu et al., 2024b) or compression (Wen et al., 2025). Token-reduction methods reduce visual computations through action-aware dynamic pruning (Pei et al., 2025), self-speculative pruning (Wang et al., 2025a), training-free token compression (Yang et al., 2025), or interaction-aligned token selection (Cheng et al., 2026). Quantization reduces memory overhead and arithmetic precision (Park et al., 2024; Wang et al., 2025b), while action-space optimization accelerates decoding by leveraging compact action tokenization (Pertsch et al., 2025) and parallel action-chunk prediction (Kim et al., 2025). VLA caching is a complementary optimization that leverages cross-frame redundancy to cache partial visual features. VLA-Cache adopts a predefined text-to-vision attention criterion for selecting tokens to cache, while LAC learns its caching policy via task supervision yet still leverages RAFT-small optical flow as an external cue (Xu et al., 2025; Wei et al., 2026). However, both methods exhibit critical limitations: reliance on predefined or externally computed proxy cues and insufficient temporal modeling capabilities.

3 METHODOLOGY

3.1 PRELIMINARY: KV CACHING FOR VLA TOKEN REUSE

KV caching accelerates Transformer inference by retaining the key and value states of tokens to avoid redundant computations. For a Transformer layer l and an input sequence $\mathbf{X}_t^l$ at control step t, the projected query $\mathbf{Q}_t^l$, key $\mathbf{K}_t^l$, value $\mathbf{V}_t^l$, and attention output $\mathbf{O}_t^l$ are:

$$\mathbf{Q}_t^l = \mathbf{X}_t^l W_Q^l, \qquad \mathbf{K}_t^l = \mathbf{X}_t^l W_K^l, \qquad \mathbf{V}_t^l = \mathbf{X}_t^l W_V^l, \tag{1}$$

$$\mathbf{O}_t^l = \mathrm{softmax}\Big(\mathbf{Q}_t^l (\mathbf{K}_t^l)^\top / \sqrt{d_l}\Big)\, \mathbf{V}_t^l, \tag{2}$$

where d_l is the attention head dimension. In standard autoregressive decoding, previous K/V states are stored and concatenated with those of newly generated tokens, eliminating repeated computations on historical context tokens (Touvron et al., 2023; Jiang et al., 2023; Brown et al., 2020).

VLA inference exhibits a similar form of redundancy across successive control steps (see Fig. 1). Let $\mathbf{X}_t = \{\mathbf{x}_t^{(i)}\}_{i=1}^N$ denote the visual tokens from the observation at step t. We divide these tokens into an active set $\mathcal{A}_t$, whose states are recomputed from the current observation, and a cached set $\mathcal{C}_t$, whose previous states are directly reused. For a layer l and token i, the updated cache states are:

$$\big(\mathbf{k}_t^{l,i}, \mathbf{v}_t^{l,i}\big) = \begin{cases} \big(\widehat{\mathbf{k}}_t^{l,i}, \widehat{\mathbf{v}}_t^{l,i}\big), & i \in \mathcal{A}_t, \\ \big(\mathbf{k}_{t-1}^{l,i}, \mathbf{v}_{t-1}^{l,i}\big), & i \in \mathcal{C}_t, \end{cases} \qquad \mathcal{A}_t \cap \mathcal{C}_t = \varnothing, \tag{3}$$

where $(\widehat{\mathbf{k}}_t^{l,i}, \widehat{\mathbf{v}}_t^{l,i})$ are computed states. Critically, cached tokens are not evicted from the context unlike token-pruning schemes (Chen et al., 2024; Zhang et al., 2024; Bolya et al., 2022; Cao et al., 2023; 2024); their stored K/V states remain accessible to all queries for attention computation. Consequently, computation is reduced by only updating a subset of visual tokens, while the model retains access to the full visual context. This temporal variant of KV caching is particularly well-suited for robotic manipulation tasks: static background regions can be reused across steps, whereas dynamic and task-relevant regions are refreshed at every control step (Xu et al., 2025; Wei et al., 2026).

3.2 OVERVIEW

In this section, we introduce **MemCache**, a memory-augmented adaptive token caching framework. As illustrated in Fig. 2, we first introduce a bank of learnable queries that extract token-level importance features without relying on hand-crafted proxy signals in Sec. 3.3.1. Next, Sec. 3.3.2 presents a recurrent visual memory state that retrieves history-aware evidence from current observations to correct query-based caching decisions. During training, we employ the Gumbel-Softmax straight-through estimator (STE) paired with an annealed budget objective to render the training process differentiable, as detailed in Sec. 3.4.1. Sec. 3.4.2 further introduces our linear-warmup schedule with a novel memory-consistency objective, which gradually integrates memory-conditioned corrections into query-based caching for joint optimization.

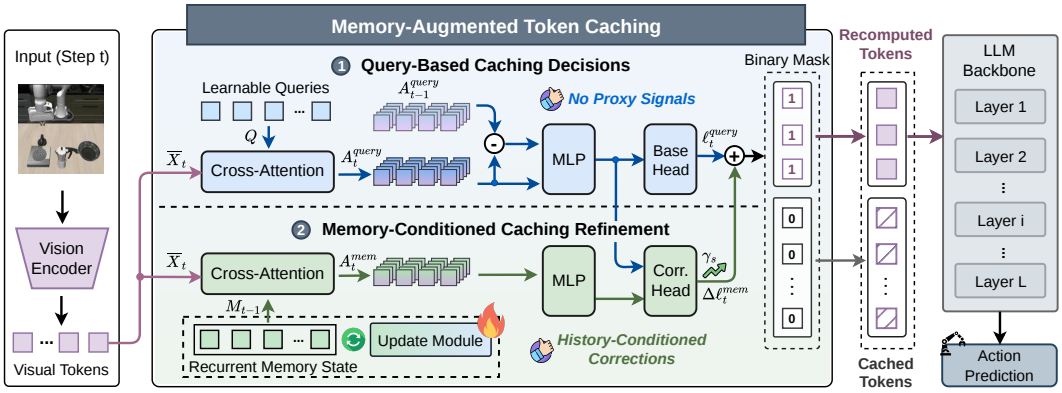

Figure 2: Overview of our memory-augmented token caching, which employs ① a bank of learnable queries to produce caching decisions free of hand-crafted proxy signals, alongside ② a recurrent visual memory state that retrieves history-aware evidence and provides corrections to caching decisions. Note that the update module refreshes the memory state to accumulate useful historical context for better subsequent caching performance (see bottom-left part and detailed in Fig. 3).

3.3 Memory-Augmented Token Caching

3.3.1 Query-Based Caching Decisions

As illustrated in Step ① of Fig. 2, we first use K learnable query anchors $\mathbf{Q}_{\text{learn}} \in \mathbb{R}^{K\times d}$ to summarize the current visual input, where d denotes the embedding dimension. After layer normalization and multi-head projection, the query-to-visual attention is:

$$\mathbf{A}_t^{\text{query}} = \text{softmax}\Big(\overline{\mathbf{Q}}_{\text{learn}}\,\overline{\mathbf{X}}_t^{\top}/\sqrt{d_h}\Big), \qquad \mathbf{A}_t^{\text{query}} \in \mathbb{R}^{H\times K\times N}, \tag{4}$$

where H and d_h are the number and dimension of attention heads, and N is the number of projected visual tokens in the current observation. $\overline{\mathbf{Q}}_{\text{learn}}$ and $\overline{\mathbf{X}}_t$ represent the normalized and head-partitioned query and visual representations, respectively. Since queries are reused at every control step, their attention outputs serve as a consistent reference across timesteps. Notably, in addition to the instantaneous attention map $\mathbf{A}_t^{\text{query}}$, which captures each token's momentary relevance to the learned visual queries, we incorporate its delta $|\mathbf{A}_t^{\text{query}} - \mathbf{A}_{t-1}^{\text{query}}|$ to highlight newly changing image regions. By fusing these two signals, the gate gains access to both absolute saliency and temporal variation, enabling it to distinguish persistently important tokens from those whose importance has recently risen. Accordingly, the input feature is:

$$\boldsymbol{\phi}_{t,i}^{\text{query}} = \text{concat}(\mathbf{A}_{t,i}^{\text{query}}, |\mathbf{A}_{t,i}^{\text{query}} - \mathbf{A}_{t-1,i}^{\text{query}}|). \tag{5}$$

An MLP f_{query} then maps this feature to a hidden representation $\mathbf{h}_{t,i}^{\text{query}}$ and finally produces decision logits $\boldsymbol{\ell}_{t,i}^{\text{query}} \in \mathbb{R}^2$ for the binary choice between *reuse* and *recompute* via f_{base}:

$$\mathbf{h}_{t,i}^{\text{query}} = f_{\text{query}}\boldsymbol{\phi}_{t,i}^{\text{query}}, \qquad \boldsymbol{\ell}_{t,i}^{\text{query}} = f_{\text{base}}(\mathbf{h}_{t,i}^{\text{query}}). \tag{6}$$

Unlike pixel- or motion-based scoring heuristics, these features together with their selection rule are jointly learned under the control objective, yielding more accurate caching decisions.

3.3.2 Memory-Conditioned Caching Refinement

Despite its effectiveness, the query-based decision gate described above relies solely on the instantaneous query-to-visual attention feature and its delta, yielding only short-range contextual information. To remedy this, we maintain a compact recurrent visual state $\mathbf{M}_t \in \mathbb{R}^{K\times d}$ to accumulate task-relevant evidence across control steps. This preserves the relevance of regions with minimal instantaneous change and mitigates gate oscillations induced by transient visual noise. As shown in Step ② of Fig. 2, at step t, the memory state from the previous step $\mathbf{M}_{t-1}$ queries the current visual tokens to obtain a history-conditioned relevance map:

$$\mathbf{A}_t^{\text{mem}} = \text{softmax}\Big(\overline{\mathbf{M}}_{t-1}\,\overline{\mathbf{X}}_t^{\top}/\sqrt{d_h}\Big), \tag{7}$$

where $\overline{\mathbf{M}}_{t-1}$ denotes the projected and normalized memory state. We use this signal as a correction rather than directly replacing the previous query-based gate:

$$\mathbf{h}_{t,i}^{\text{mem}} = f_{\text{mem}}\mathbf{A}_{t,i}^{\text{mem}}, \tag{8}$$

$$\Delta\boldsymbol{\ell}_{t,i}^{\text{mem}} = f_{\text{corr}}\left([\text{stopgrad}(\mathbf{h}_{t,i}^{\text{query}}) \,\|\, \mathbf{h}_{t,i}^{\text{mem}}]\right), \qquad \boldsymbol{\ell}_{t,i} = \boldsymbol{\ell}_{t,i}^{\text{query}} + \gamma_s \Delta\boldsymbol{\ell}_{t,i}^{\text{mem}}. \tag{9}$$

The correction head f_{corr} receives the stop-gradient query-based hidden feature $\mathbf{h}_{t,i}^{\text{query}}$ and the memory feature $\mathbf{h}_{t,i}^{\text{mem}}$, allowing it to compute corrections $\Delta\boldsymbol{\ell}_{t,i}^{\text{mem}}$ conditioned on base decisions $\boldsymbol{\ell}_{t,i}^{\text{query}}$ defined in Eq. (6). Here, $\gamma_s \in [0, 1]$ is a warm-up coefficient that progressively increases during training, and the final layer of f_{corr} is zero-initialized. Together, these two designs ensure the initial gate output $\boldsymbol{\ell}_{t,i}$ exactly matches the base $\boldsymbol{\ell}_{t,i}^{\text{query}}$ and enable history-aware corrections to emerge gradually, aiming to minimize the influence of $\Delta\boldsymbol{\ell}_{t,i}^{\text{mem}}$ on gate decisions during the unstable early training phase. Appendix. A.2 summarizes the complete inference pipeline of our token caching.

Recurrent State Update. To preserve historical context while incorporating current visual evidence for providing better caching decisions in subsequent control steps, we update the recurrent visual memory state (see the bottom-left part of Fig. 2). Specifically, we first construct a compact observation summary based on the query-to-visual attention map at the current step $\mathbf{A}_t^{\text{query}}$:

$$\mathbf{S}_t = W_s(\mathbf{A}_t^{\text{query}}\overline{\mathbf{X}}_t), \tag{10}$$

where W_s denotes the learnable projection weights. As depicted in Fig. 3, given $\mathbf{M}_{t-1}$, the update module f_{upd} predicts a preserve gate $\mathbf{a}_t$ and a candidate update $\widetilde{\mathbf{M}}_t$:

$$[\mathbf{a}_t, \widetilde{\mathbf{M}}_t] = f_{\text{upd}}([\text{LN}(\mathbf{M}_{t-1}) \,\|\, \text{LN}(\mathbf{S}_t)]), \quad \mathbf{z}_t = \sigma(\mathbf{a}_t),$$
$$\mathbf{M}_t = \mathbf{z}_t \odot \mathbf{M}_{t-1} + (1 - \mathbf{z}_t) \odot \tanh(\widetilde{\mathbf{M}}_t). \tag{11}$$

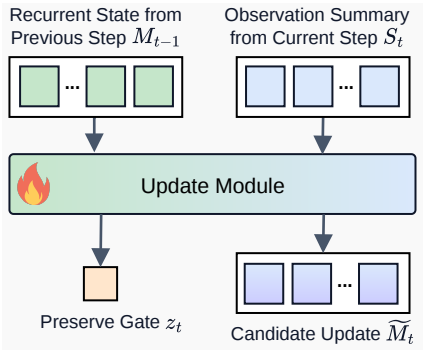

Figure 3: Illustrating state update to aggregate useful historical context.

Sigmoid $\sigma()$ and $\tanh()$ are applied element-wise, so $\mathbf{z}_t \in \mathbb{R}^{K\times d}$ independently controls the retention strength for each memory entry: a large value of $\mathbf{z}_t$ favors the preservation of historical entries, while a small value encourages replacement by the candidate state derived from the current observation. The resulting updated state $\mathbf{M}_t$ is propagated to the subsequent control step $t + 1$ and consumed by the memory-conditioned gate described above. Note that for the initial observation with no prior memory state available, $\mathbf{M}_0$ is directly initialized from $\mathbf{S}_0$.

3.4 Training Strategies

3.4.1 Differentiable Caching Decisions with Budget Annealing

The caching decisions are discrete, prohibiting direct backpropagation through the hard mask. To solve this issue, we adopt the Gumbel-Softmax STE (Jang et al., 2017; Bengio et al., 2013). Given the two-class logits $\boldsymbol{\ell}_{t,i}$, we sample Gumbel noise $\boldsymbol{\epsilon}_{t,i} \sim \text{Gumbel}(0, 1)$ and construct a perturbed soft categorical distribution:

$$\widetilde{\mathbf{p}}_{t,i} = \text{softmax}((\boldsymbol{\ell}_{t,i} + \eta_s\boldsymbol{\epsilon}_{t,i})/\tau_s). \tag{12}$$

Here $\tau_s{>}0$ is the temperature at optimization step s, and η_s controls the scale of the Gumbel perturbation. A larger η_s produces a smoother probability distribution, while a smaller one makes the distribution closer to a discrete decision. The forward pass converts $\widetilde{\mathbf{p}}_{t,i}$ into a one-hot decision:

$$\mathbf{y}_{t,i}^{\text{hard}} = \text{one-hot}\left(\arg\max_{c\in\{0,1\}} \widetilde{\mathbf{p}}_{t,i}\right). \tag{13}$$

During backpropagation, the STE follows the corresponding Gumbel-perturbed soft probability:

$$\mathbf{y}_{t,i} = \mathbf{y}_{t,i}^{\text{hard}} - \text{stopgrad}(\widetilde{\mathbf{p}}_{t,i}) + \widetilde{\mathbf{p}}_{t,i}, \qquad g_{t,i} = \mathbf{y}_{t,i}^{\text{recompute}}. \tag{14}$$

This "hard-forward, soft-backward" construction enables discrete caching token selection while preserving gradients. We anneal both the temperature τ_s and exploration noise η_s: training starts with smoother gradients and stochastic exploration, progressively converging to deterministic inference.

To further constrain the computational budget, we regularize the mean recomputation ratio:

$$\mathcal{L}_{\text{budget}} = \left(\frac{1}{N} \sum_{i=1}^{N} g_{t,i} - r_s \right)^2 , \tag{15}$$

where r_s is the target ratio at optimization step s. Rather than imposing an aggressive budget from the beginning, we linearly decay the target from a conservative starting ratio r_0 to the final ratio $r_\star$:

$$r_s = r_0 + (r_\star - r_0) \min(s/S_{\text{budget}}, 1) . \tag{16}$$

S_{budget} is the number of optimization steps allocated to this budget-annealing phase. This schedule first stabilizes action prediction by relying predominantly on fresh features, then gradually raises the cache pressure to encourage higher cache utilization with boosted inference efficiency.

3.4.2 Linear-Warmup Training with Memory Consistency

Memory Consistency Objective. The memory state is built to preserve *temporally-stable* and *salient* visual evidence. However, without explicit supervision, it tends to drift toward short-lived fluctuations. Thus, we introduce a memory-consistency objective that aligns the history-conditioned memory-to-visual attention map $\mathbf{A}_t^{\text{mem}}$ with a stable target derived from the consecutive query-to-visual attention map $\mathbf{A}_{t/t-1}^{\text{query}}$. Specifically, we first construct a stability weight $\mathbf{W}_t$ to locate regions exhibiting frame-level *stability*, and a cross-frame consensus target $\mathbf{T}_t$ for capturing persistent *saliency*:

$$\mathbf{W}_t = \exp\!\left(-\beta |\mathbf{A}_t^{\text{query}} - \mathbf{A}_{t-1}^{\text{query}}|\right), \qquad \mathbf{T}_t = \sqrt{\mathbf{A}_t^{\text{query}} \odot \mathbf{A}_{t-1}^{\text{query}}}. \tag{17}$$

$\mathbf{W}_t$ approaches 1 for temporally-stable regions and decays toward 0 when temporal characteristics shift. Here $\beta > 0$ controls sensitivity to temporal variations. The target entry $\mathbf{T}_t$ is defined as the geometric mean of the two attention maps. This symmetric construction produces a high target value only if a token exhibits saliency across both observations, thereby mitigating the influence of instantaneous saliency. We then restrict our supervision to stable regions by reweighting both $\mathbf{A}_t^{\text{mem}}$ and $\mathbf{T}_t$ with $\mathbf{W}_t$, and normalize each into a distribution over N visual tokens:

$$\mathbf{P}_t^{\text{mem}} = \text{Norm}(\mathbf{A}_t^{\text{mem}} \odot \mathbf{W}_t), \qquad \mathbf{P}_t^{\text{tgt}} = \text{Norm}(\mathbf{T}_t \odot \mathbf{W}_t). \tag{18}$$

$\mathbf{P}_t^{\text{mem}}$ represents the token distribution retrieved by the recurrent memory, while $\mathbf{P}_t^{\text{tgt}}$ denotes the target distribution for regions that are both salient and stable across adjacent observations. Applying the same $\mathbf{W}_t$ to both distributions constrains their comparison to stable and salient cross-frame evidence. We then minimize their Jensen-Shannon (JS) divergence:

$$\mathcal{L}_{\text{cons}} = \frac{1}{HK} \sum_{h=1}^{H} \sum_{k=1}^{K} D_{\text{JS}} \left(\mathbf{P}_{t,h,k,:}^{\text{mem}} \middle\| \mathbf{P}_{t,h,k,:}^{\text{tgt}} \right) . \tag{19}$$

The loss is averaged over all H attention heads and K anchor queries. We stop gradients through $\mathbf{W}_t$ and $\mathbf{T}_t$, treating them as fixed targets. Consequently, $\mathcal{L}_{\text{cons}}$ solely trains the memory path to recover stable cross-frame evidence while leaving the query-based target unchanged.

Linear-Warmup Training Strategy. In summary, we optimize our MemCache with the following objective:

$$\mathcal{L}_t = \mathcal{L}_{\text{action},t} + \lambda_{\text{budget}} \mathcal{L}_{\text{budget},t} + \lambda_{\text{cons}} \mathcal{L}_{\text{cons},t}. \tag{20}$$

$\mathcal{L}_{\text{action},t}$ is the VLA action-prediction loss for task performance optimization, $\mathcal{L}_{\text{budget},t}$ denotes the computational budget loss (Eq. (16)) that promotes caching to improve inference efficiency, and $\mathcal{L}_{\text{cons},t}$ represents the memory-consistency loss (Eq. (19)) for stabilizing recurrent memory training. The coefficients λ_{budget} and λ_{cons} jointly balance task accuracy, computational costs, and temporal memory stability. Note that, as shown in Fig. 4, λ_{cons} is a linearly decaying coefficient that gradually decreases throughout training. This design stabilizes the optimization of recurrent memory states and facilitates reliable memory initialization. It complements the warm-up coefficient γ_s in Eq. (9), which progressively increases to incorporate memory-conditioned corrections into final gate decisions so that they are more strongly supervised by the action loss. Together, these two scheduling strategies enable the memory module to undergo robust training in the early stage and align more accurately with real action signals in the later stage.

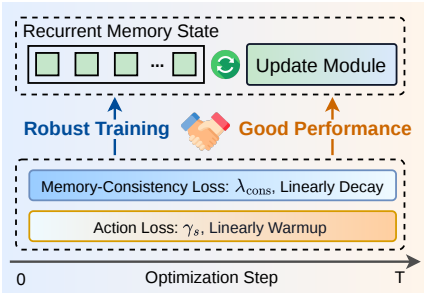

Figure 4: Illustrating training schedule.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Compared Methods. To evaluate the effectiveness of our MemCache, we compare it with a broad range of token pruning and caching baselines. We first consider representative token-level acceleration methods originally developed for vision-language models (VLMs), including SparseVLM (Zhang et al., 2024) and FastV (Chen et al., 2024). We further compare against four VLA-tailored token pruning approaches: VLA-ADP (Pei et al., 2025), SpecPrune-VLA (Wang et al., 2025a), EfficientVLA (Yang et al., 2025), and IAprune (Cheng et al., 2026), as well as two strong VLA-dedicated caching methods, VLA-Cache (Xu et al., 2025) and LAC (Wei et al., 2026). We evaluate our method on three representative VLA models covering different action generation paradigms: OpenVLA formulates robotic control as next-token prediction by discretizing continuous actions into action tokens (Kim et al., 2024); OpenVLA-OFT employs a parallel continuous-action decoding head and supports action chunking to achieve a higher control frequency (Kim et al., 2025); and CogACT integrates a diffusion-based action module that generates continuous and temporally coherent action sequences for precise robotic control (Li et al., 2024a).

Evaluation Metrics. We evaluate our method in both task performance and inference efficiency. For task performance, we report the success rate, defined as the fraction of evaluation episodes where the robot completes the manipulation task. For efficiency, we quantify the computational costs via floating-point operations (FLOPs) and latency. Unless otherwise specified, latency measurements are collected on a single NVIDIA RTX 4090 GPU after sufficient warm-up iterations, following the evaluation protocols in previous works (Xu et al., 2025; Wei et al., 2026) to ensure fair comparisons. Detailed optimization recipes are provided in Appendix A.4.

4.2 EVALUATION BENCHMARK

LIBERO. LIBERO (Liu et al., 2024a) is a simulated tabletop-manipulation benchmark to evaluate robotic policies across four distinct task suites. The *Spatial* suite probes generalization over object spatial relations; *Object* targets generalization to unseen object instances; *Goal* varies the target task outcome; and *Long* assembles primitive skills into long-horizon instructions. Each suite comprises ten tasks. We report the mean success rate averaged across all tasks and repeated episodes.

SIMPLER. SIMPLER (Li et al., 2024c) provides a reproducible simulation protocol for studying manipulation policies intended for real-robot deployment. We follow the commonly used Google robot-arm setup and evaluate its four tasks under both *Visual Matching* and *Variant Aggregation*. Visual Matching aligns simulated rendering with corresponding real-world observations, while Variant Aggregation averages performance across controlled variations of appearance and environment. This benchmark complements LIBERO by examining the robustness of efficient inference under visual and scene variations.

4.3 EXPERIMENTAL RESULTS

Main Results on LIBERO. As shown in Table 1, our method yields consistent improvements in the trade-off between task performance and inference efficiency over all baselines, validating its effectiveness. Specifically, compared with the vanilla OpenVLA and OpenVLA-OFT, we achieve on average $\uparrow \mathbf{3.1}\%$ and $\uparrow \mathbf{1.3}\%$ higher success rates, alongside $\uparrow \mathbf{1.87}\times$ and $\uparrow \mathbf{1.83}\times$ speedups, respectively. This suggests that well-designed token-caching policies enable the allocation of computations to more critical regions, thereby mitigating input noise to boost overall performance. Furthermore, against strong VLA caching baselines, our approach obtains an average $\uparrow \mathbf{1.2}\%$ higher success rate with a $\uparrow 1.07\times$ speedup over LAC (Wei et al., 2026) on OpenVLA, as well as a $\uparrow \mathbf{0.7}\%$ higher success rate and a $\uparrow \mathbf{1.45}\times$ speedup over VLA-Cache (Xu et al., 2025) for OpenVLA-OFT, further confirming our superiority. Note that our FLOPs reduction over caching baselines does not fully translate into practical speedup on OpenVLA relative to OpenVLA-OFT. For example, we achieve $\downarrow \mathbf{44.2}\%$ fewer FLOPs but merely $1.07\times$ practical speedup over LAC for OpenVLA, versus a $\downarrow \mathbf{46.4}\%$ FLOPs reduction and a corresponding $\mathbf{1.45}\times$ speedup over VLA-Cache for OpenVLA-OFT. This is because OpenVLA contains 50% fewer visual tokens than OpenVLA-OFT, making the model prone to becoming *memory-bound* on GPUs under higher caching ratios and thereby hindering the theoretical computational benefits. Appendix A.3 provides a detailed efficiency analysis regarding the overhead caused by our gate decision and the savings introduced by our token caching.

Table 1: Results on the LIBERO benchmark. Our method achieves better trade-offs between task performance and inference efficiency across all baselines.

Method	Success Rate ↑					FLOPs (T) ↓	Latency (ms) ↓
	Spatial	Object	Goal	Long	Average		
OpenVLA	84.4%	86.6%	75.6%	53.2%	75.0%	1.864	51.91
SparseVLM	79.8%	67.0%	72.6%	39.4%	64.7%	1.407	83.39
FastV	83.4%	84.0%	74.2%	51.6%	73.3%	1.864	53.28
VLA-Cache	83.8%	85.8%	76.4%	52.8%	74.7%	1.355	31.83
LAC	85.6%	86.2%	76.6%	**59.2%**	76.9%	1.392	29.51
Ours	**87.6%**	**87.6%**	**78.2%**	59.0%	**78.1%**	**0.777**	**27.69**
OpenVLA-OFT	97.8%	97.6%	97.6%	94.2%	96.8%	4.013	79.05
VLA-Cache	98.3%	97.5%	**98.3%**	95.4%	97.4%	3.097	62.59
Ours	**99.2%**	**98.8%**	98.0%	**96.2%**	**98.1%**	**1.660**	**43.23**

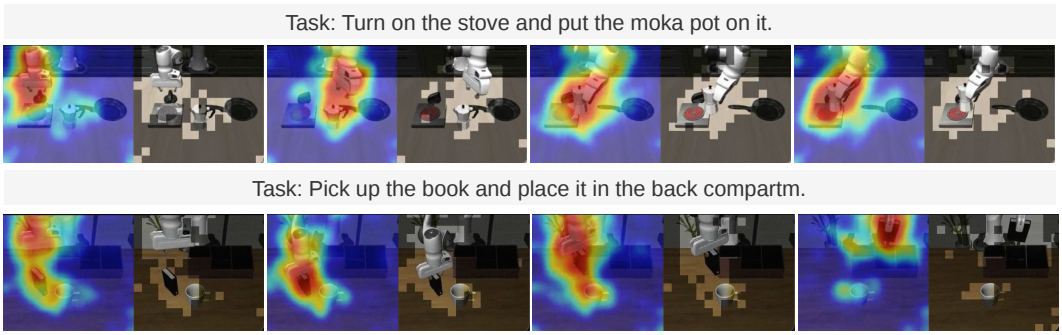

Figure 5: Visualization of our caching decisions on the LIBERO *Long* suite with OpenVLA. The left columns visualize heatmaps of our gate outputs, where greater red intensity corresponds to higher confidence for token recomputation and blue denotes tokens eligible for caching. The right columns illustrate the actual caching outcomes, with cached tokens masked for visually intuitive comparison.

Comparison with Pruning Methods. As shown in Table 2, where the best results are marked in **bold** and the second-best results are underlined, our method consistently outperforms pruning baselines in trade-offs between task performance and computational efficiency. Specifically, we achieve an average $\uparrow 1.0\%$ higher success rate at an even lower computation ratio compared with IAprune (Cheng et al., 2026). This demonstrates the superiority of our learnable memory-augmented caching framework and the benefits of caching: preserving KV states of static tokens for attention computation to supply richer contextual information.

Table 2: Comparison with pruning methods on the LIBERO benchmark using OpenVLA-OFT.

Method	Success Rate (%) ↑					Pruning/ Caching Ratio↑
	Spatial	Object	Goal	Long	Avg.	
OpenVLA-OFT	97.8	97.6	97.6	94.2	96.8	–
VLA-ADP	**99.4**	98.0	96.4	91.2	96.3	0.50
VLA-ADP	97.6	98.4	97.4	84.2	94.4	0.70
SpecPrune-VLA	97.4	95.8	97.7	93.4	96.1	**0.80**
EfficientVLA	96.5	91.1	96.0	72.1	88.9	0.70
IAprune	97.3	**99.1**	**98.2**	95.2	97.5	0.50
IAprune	96.6	98.8	98.0	94.8	97.1	0.70
Ours	99.2	98.8	98.0	**96.2**	**98.1**	0.75

Qualitative Results. Fig. 5 (left columns) visualizes heatmaps generated by our gate outputs: red marks tokens identified as important by the gate and requiring recomputation, whereas blue denotes unimportant tokens eligible for caching. The right columns illustrate the corresponding caching decisions. As observed, the focus of our caching gate closely follows the robotic arm and salient objects within each current step, which directly shapes action prediction and thus task performance. For instance, in the first row of Fig. 5, the assigned task is "Turn on the stove and put the moka pot on it." Beyond static background regions, our caching gate disregards the irrelevant frying pan. Accordingly, it allocates more focus to the stove in early steps, the moka pot in intermediate steps, and both objects in later steps. This verifies that our caching gate dynamically and accurately identifies critical visual elements across timesteps. As a result, it properly directs computations to salient regions for boosting task performance, while suppressing irrelevant tokens to improve inference efficiency. For more comprehensive analysis and insights, we provide qualitative results across *all*

four task suites from the LIBERO benchmark in Appendix A.7. These visualizations validate our effectiveness under four distinct scenarios: recognizing varied spatial locations, identifying unseen objects, following task instructions, and handling long-horizon tasks.

Ablation Studies. To validate each component in MemCache, we conduct ablation studies. As shown in Table 3, even at a 70% caching ratio, the query-based caching decisions defined in Eq. (6) already delivers higher success rates than both base OpenVLA and OpenVLA-OFT baselines when evaluated on the LIBERO *Spatial* task. This confirms the effectiveness of direct end-to-end optimization of learnable signals under action supervision, without relying on any hand-crafted proxy ones. Incorporating our recurrent memory state in Eq. (9) further boosts task performance at the same caching ratio, verifying its ability to refine caching decisions through history-driven corrections. Moreover, combining our linear-warmup training schedule with the proposed memory consistency loss described in Sec. 3.4.2 brings additional $\uparrow\mathbf{1.4}\%$ and $\uparrow 0.2\%$ gains in success rate at comparable caching ratios on OpenVLA and OpenVLA-OFT, respectively. This stems from the gradual activation of memory-conditioned corrections to facilitate joint optimization. Note that as detailed in Appendix A.3, our gate introduces merely $\mathbf{0.3}\%\sim\mathbf{0.6}\%$ overhead over full models, leading to negligible computational cost. Additional ablation results are provided in Appendices A.5 and A.6. Appendix A.7 further includes qualitative comparisons for intuitive explanations.

Table 3: Ablation studies on LIBERO *Spatial*.

Method	OpenVLA			OpenVLA-OFT		
	Success Rate ↑		Ratio ↑	Success Rate ↑		Ratio ↑
Base Model	84.4%	–	–	97.8%	–	–
+ Queries	85.4%	↑1.0%	**0.70**	98.6%	↑0.8%	0.70
+ Memory	86.2%	↑1.8%	**0.70**	99.0%	↑1.2%	0.70
+ Warmup	**87.6%**	**↑ 3.2%**	**0.70**	**99.2%**	**↑ 1.4%**	**0.80**

Table 4: Compared with existing caching methods on the SIMPLER benchmark, where best results are in **bold** and second-best results are underlined.

SIMPLER	Method	Success Rate ↑					FLOPs (T) ↓	Latency (ms) ↓
		PickCan	MoveNear	Drawer	DrawerApple	Average		
Visual Matching	CogACT	91.3%	**85.0%**	71.8%	50.9%	74.8%	1.847	54.29
	VLA-Cache	92.0%	83.3%	70.5%	51.6%	74.4%	1.496	39.63
	LAC	92.3%	84.2%	72.7%	**52.8%**	**75.5%**	1.511	37.82
	Ours	**93.1%**	84.9%	**72.8%**	50.9%	75.4%	**1.035**	**31.71**
Visual Aggregation	CogACT	89.6%	80.8%	28.3%	46.6%	61.3%	1.807	53.54
	VLA-Cache	91.7%	79.3%	**32.5%**	45.8%	62.3%	1.493	39.11
	LAC	92.1%	**81.5%**	31.2%	47.1%	63.0%	1.506	37.90
	Ours	**92.7%**	80.8%	31.5%	**48.1%**	**63.3%**	**1.244**	**34.23**

Main Results on SIMPLER. To further verify the robustness and generalizability of MemCache, we conduct experiments with the CogACT model on the SIMPLER benchmark, with results summarized in Table 4. Specifically, compared with the vanilla CogACT, we achieve on average $\uparrow\mathbf{0.6}\%$ and $\uparrow\mathbf{2.0}\%$ higher success rates, alongside $\uparrow\mathbf{1.71}\times$ and $\uparrow\mathbf{1.56}\times$ speedups on the visual matching and aggregation settings, respectively. This demonstrates that our caching mechanism selectively allocates computation resources to critical tokens (see Appendix A.7 for visualizations), thereby suppressing noisy input features to boost task performance. Moreover, compared with the caching baseline LAC, our method achieves comparable success rates ($\downarrow 0.1\%$ and $\uparrow 0.3\%$) while delivering lower FLOPs ($\downarrow\mathbf{31.5}\%$ and $\downarrow\mathbf{17.4}\%$) as well as higher speedups ($\uparrow 1.19\times$ and $\uparrow 1.11\times$) for visual matching and aggregation settings, respectively, further validating our advantages.

5 Conclusion

This work presents MemCache, a learnable memory-augmented caching framework for efficient Vision-Language-Action (VLA) inference. It incorporates three core components: learnable queries that enable end-to-end token-caching optimization directly under the control objective without hand-crafted proxy signals, recurrent visual memory for accumulating temporal information, and a linear-warmup training schedule that coordinates the above two components for joint optimization. Extensive evaluations validate that MemCache reduces inference latency and computational overhead with even boosted manipulation performance. However, MemCache is constrained by a global uniform caching ratio that ignores layer-level and task-phase variations, which motivates future work on adaptive budget allocation and a unified pipeline integrating token pruning and caching for better performance-efficiency trade-offs. More details are provided in Appendix A.1.

AI USE STATEMENT

In this work, we employed generative AI tools for code revision, manuscript editing, and literature summarization. We did not used them to gather data, alter results, or derive scientific conclusions. All AI-assisted code and text were carefully reviewed and revised by the authors. Citations and related work descriptions were manually validated against original papers, and all reported numbers come from our own experiments. The authors take complete responsibility for the final manuscript, including all text, claims, results, and artifacts generated with generative AI support.

REPRODUCIBILITY STATEMENT

We present the methodological and experimental details necessary for reproducing our MemCache in the manuscript. Specifically, Sec. 3 describes our methodology in detail, including the query-based caching decisions, memory-conditioned caching refinement, differentiable caching decisions with budget annealing, and the linear-warmup training strategy equipped with the memory-consistency objective. Additionally, Appendix A.2 outlines the full inference pipeline, covering caching-gate decisions, full-context KV construction, and attention computation. Appendix A.4 elaborates on the experimental configurations needed to replicate our experiments, such as architectural hyperparameters, loss coefficients, and training recipes. To further facilitate reproducibility, we will release the complete codebase for training and evaluation along with pretrained checkpoints upon paper acceptance.

REFERENCES

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

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-VL: A versatile vision-language model for understanding, localization, text reading, and beyond. *arXiv preprint arXiv:2308.12966*, 2023.

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

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

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.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Jianjian Cao, Peng Ye, Shengze Li, Chong Yu, Yansong Tang, Jiwen Lu, and Tao Chen. MADTP: Multimodal alignment-guided dynamic token pruning for accelerating vision-language transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15710–15719, 2024.

Qingqing Cao, Bhargavi Paranjape, and Hannaneh Hajishirzi. PuMer: Pruning and merging tokens for efficient vision language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 12890–12903, 2023.

Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*, volume 15139 of *Lecture Notes in Computer Science*, pp. 19–35. Springer, 2024.

Jintao Cheng, Weibin Li, Haozhe Wang, Gang Wang, Yipu Zhang, Xiaoyu Tang, Jin Wu, Xieyuanli Chen, Yunhui Liu, and Wei Zhang. Training-free interaction-aligned visual token pruning for efficient embodied manipulation. *arXiv preprint arXiv:2603.22991*, 2026.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyi Li, Pascale N. Fung, and Steven C. H. Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 49250–49267, 2023.

Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, Guang Shi, and Haoqi Fan. Emerging properties in unified multimodal pretraining. *arXiv preprint arXiv:2505.14683*, 2025.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, et al. PaLM-E: An embodied multimodal language model. In *International Conference on Machine Learning*, pp. 8469–8488, 2023.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, et al. Openvla: An open-source vision-language-action model. In *Conference on Robot Learning*, pp. 2679–2713, 2024.

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

Junnan Li, Dongxu Li, Silvio Savarese, and Steven C. H. Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International Conference on Machine Learning*, pp. 19730–19742, 2023.

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

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

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024c.

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, 2024a.

Jiaming Liu, Mengzhen Liu, Zhenyu Wang, Pengju An, Senqiao Yang, et al. Robomamba: Efficient vision-language-action model for robotic reasoning and manipulation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 40085–40110, 2024b.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, et al. RDT-1B: A diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, et al. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems*, 2024.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6892–6903. IEEE, 2024.

Seongmin Park, Hyungmin Kim, Wonseok Jeon, Juyoung Yang, Byeongwook Jeon, et al. Quantization-aware imitation learning for resource-efficient robotic control. *arXiv preprint arXiv:2412.01034*, 2024.

Xiaohuan Pei, Yuxing Chen, Siyu Xu, Yunke Wang, Yuheng Shi, and Chang Xu. Action-aware dynamic pruning for efficient vision-language-action manipulation. *arXiv preprint arXiv:2509.22093*, 2025.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, et al. FAST: Efficient action tokenization for vision-language-action models. *arXiv preprint arXiv:2501.09747*, 2025.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, et al. LLaMA 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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. *arXiv preprint arXiv:2509.05614*, 2025a.

Hongyu Wang, Chuyan Xiong, Ruiping Wang, and Xilin Chen. Bitvla: 1-bit vision-language-action models for robotics manipulation. *arXiv preprint arXiv:2506.07530*, 2025b.

Yujie Wei, Jiahan Fan, Jiyu Guo, Ruichen Zhen, Rui Shao, Xiu Su, Zeke Xie, and Shuo Yang. Learning to accelerate vision-language-action models through adaptive visual token caching. *arXiv preprint arXiv:2602.00686*, 2026.

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

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

Yantai Yang, Yuhao Wang, Zichen Wen, Zhongwei Luo, 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.

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. *arXiv preprint arXiv:2410.04417*, 2024.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pp. 2165–2183, 2023.

A Appendix

A.1 Limitations

Despite the effectiveness of MemCache, our method has several limitations. First, we currently employ a global caching ratio that enforces uniform caching budgets across all Transformer layers and robot action steps. However, attention distributions differ substantially across layers: early layers show dispersed attention, whereas deeper layers exhibit concentrated attention and accommodate higher caching ratios. Likewise, precision requirements vary across task phases. Coarse translational movements tolerate higher caching ratios with only minor performance drops, while fine-grained grasping operations require lower ratios to maintain task accuracy. Moving forward,

we seek to develop a more flexible caching policy that dynamically allocates caching budgets on a per-layer and per-action-phase basis. Additionally, in the future, we intend to integrate token pruning and caching within a unified pipeline: pruning unimportant and static background tokens, caching important but temporarily static tokens, and reserving computations exclusively for critical and dynamically changing visual tokens. This hybrid design may further improve the performance–efficiency trade-off for VLA robotic policies.

A.2 Inference Pipeline

At each control step t, the vision encoder and projector output N visual tokens $\mathbf{X}_t \in \mathbb{R}^{N\times d}$, where d denotes the hidden dimension. Our memory-augmented caching gate first computes query-to-visual attention following Eq. (4). The resulting attention map, its temporal difference, and memory-to-visual attention map are then fed into the caching gate via Eq. (9). The gate generates a binary mask $g_{t,i}$, where $g_{t,i} = 1$ indicates a visual token requiring recomputation and $g_{t,i} = 0$ denotes a token that reuses its prior KV state. Accordingly, visual tokens are partitioned into the active set $\mathcal{A}_t = \{i : g_{t,i} = 1\}$ and the cached set $\mathcal{C}_t = \{i : g_{t,i} = 0\}$. QKV projections and MLP transformations are performed only for active visual tokens as well as the remaining non-visual tokens to reduce computations, while historical KV states of cached visual tokens are still retained in the full-length KV context to supply richer contextual information. Consequently, each active query has access to both newly-computed KV states from active positions and retained historical states from cached positions during attention computation.

Specifically, let $N_a = |\mathcal{A}_t|$ denote the number of active visual tokens and R the number of non-visual tokens, which remain active at every step. Thus, the query sequence has length $S_q = R + N_a$, whereas the KV context that reserves states for all cached tokens has length $S = R + N$. At layer l, the attention output is computed as:

$$\mathbf{O}_t^l = \mathrm{softmax}\left(\frac{\mathbf{Q}_{t,q}^l \left(\mathbf{K}_{t,\mathrm{full}}^l\right)^\top}{\sqrt{d_l}}\right)\mathbf{V}_{t,\mathrm{full}}^l, \qquad \begin{array}{l}\mathbf{O}_t^l, \mathbf{Q}_{t,q}^l \in \mathbb{R}^{S_q\times d_l},\\ \mathbf{K}_{t,\mathrm{full}}^l, \mathbf{V}_{t,\mathrm{full}}^l \in \mathbb{R}^{S\times d_l}.\end{array} \tag{21}$$

Here d_l is the dimension per attention head. The full KV context is constructed as:

$$\left(\mathbf{K}_{t,\mathrm{full}}^{l,j}, \mathbf{V}_{t,\mathrm{full}}^{l,j}\right) = \begin{cases}\left(\widehat{\mathbf{k}}_t^{l,j}, \widehat{\mathbf{v}}_t^{l,j}\right), & j \in \mathcal{I}_{\mathrm{nv}} \cup \mathcal{A}_t,\\ \left(\mathbf{k}_{t-1}^{l,j}, \mathbf{v}_{t-1}^{l,j}\right), & j \in \mathcal{C}_t.\end{cases} \tag{22}$$

Here, $\mathcal{I}_{\mathrm{nv}}$ denote the set of non-visual tokens, where $|\mathcal{I}_{\mathrm{nv}}| = R$. $\widehat{\mathbf{k}}_t^{l,j}$ and $\widehat{\mathbf{v}}_t^{l,j}$ are the freshly computed key and value states at the current step, whereas $\mathbf{k}_{t-1}^{l,j}$ and $\mathbf{v}_{t-1}^{l,j}$ are the historical states retained from the previous step. Thus, the number of queries is reduced from S to S_q to save computations, while active queries still have access to the full KV context to preserve richer information.

A.3 Efficiency Analysis

We introduce the following notation: L denotes the number of decoder layers, d the attention hidden dimension, M the MLP hidden dimension, N the number of visual tokens, and R the number of non-visual prefill tokens. As for the representative VLA models OpenVLA (Kim et al., 2024) and OpenVLA-OFT (Kim et al., 2025), both employ the LLaMA-2-7B architecture (Touvron et al., 2023) as their LLM backbone. Each model comprises $L = 32$ decoder layers, with attention hidden dimension $d = 4096$, MLP hidden dimension $M = 11008$, and 32 attention heads, yielding a per-head dimension $d_l = 128$. Specifically, OpenVLA operates with $N = 256$ visual tokens, whereas OpenVLA-OFT uses $N = 512$ visual tokens. We further let N_a represent the number of recomputed visual tokens; accordingly, $\rho = 1 - N_a/N$ defines the caching ratio.

Overhead of Memory-Augmented Gate Decision. Our query branch employs a learnable query bank $\mathbf{Q}_{\mathrm{learn}} \in \mathbb{R}^{K\times d} = \mathbb{R}^{128\times 4096}$. Following projection and head partitioning, the queries take shape $\mathbb{R}^{H_c\times K\times d_c} = \mathbb{R}^{8\times 128\times 512}$, where $d = H_c d_c = 8 \times 512 = 4096$. The projected visual tokens have shape $\mathbf{X}_t \in \mathbb{R}^{N\times d}$. The query branch then computes query-to-visual attention scores over these visual tokens following Eq. (4). The dominant cost is:

$$F_{\mathrm{query-attn}} \simeq K(H_c d_c)^2 + KN(H_c d_c). \tag{23}$$

Similarly, the recurrent memory branch projects the previous state and computes memory-to-visual scores following Eq. (7), yielding:

$$F_{\text{mem-attn}} \simeq K(H_c d_c)^2 + KN(H_c d_c). \tag{24}$$

For our memory-augmented gate module, we define the following architectural hyperparameters: $B_q = 512$, the width of the MLP that projects query-to-visual attention maps; $B_m = 256$, the width of the MLP projecting memory-to-visual attention maps; $B_r = 256$, the hidden dimension of the correction head; and $C = 2$, the number of gate classes corresponding to token reuse and recomputation. The gate cost in Eq. (9) is approximated as:

$$\begin{aligned} F_{\text{gate}} \simeq N\big[&(2H_cK)B_q + B_qC \\ &+ (H_cK)B_m + (B_q + B_m)B_r + B_rC\big], \end{aligned} \tag{25}$$

where the first row represents the query-based gate, and last row corresponds to memory-conditioned corrections.

For memory state update, which involves compact observation summary generation in Eq. (10), two projection layers for summary construction and encoding, and an update module with hidden dimension B_u in Eq. (11), the computational cost is formulated as:

$$F_{\text{state-update}} \simeq KNd + 2Kd^2 + 4KdB_u + \mathcal{O}(Kd). \tag{26}$$

Element-wise sigmoid, tanh, and residual additions are absorbed into the lower-order term $\mathcal{O}(Kd)$.

Substituting the default dimensions yields approximately 10.5 GFLOPs for $N = 256$ in OpenVLA and 11.2 GFLOPs for $N = 512$ in OpenVLA-OFT. Relative to the full costs of 1.864 TFLOPs for OpenVLA and 4.013 TFLOPs for OpenVLA-OFT, the overhead introduced by our memory-augmented gate decision is only about $\mathbf{0.6}\%$ and $\mathbf{0.3}\%$, respectively, resulting in negligible computational overhead.

Saving Introduced by Token Caching. For a full prefill of length $S = R+N$ encompassing both non-visual prefill tokens and visual tokens, the attention and MLP computations within one layer are:

$$F^{\text{full}}_{\text{attn},l} \simeq 4Sd^2 + 2S^2d, \qquad F^{\text{full}}_{\text{mlp},l} \simeq 3SdM. \tag{27}$$

The first term corresponds to the Q, K, V, and output projections; the second describes the attention score and value products; and the third accounts for the three linear transformations inside a SwiGLU block. Accordingly, $F^{\text{full}}_l = F^{\text{full}}_{\text{attn},l} + F^{\text{full}}_{\text{mlp},l}$.

For a layer that adopts token caching, with active query length $S_q = R + N_a$ and full KV context length S, the associated computations are:

$$F^{\text{cache}}_{\text{attn},l} \simeq 4S_qd^2 + 2S_qSd, \qquad F^{\text{cache}}_{\text{mlp},l} \simeq 3S_qdM. \tag{28}$$

The theoretical FLOPs saving is therefore:

$$\text{Saving} = 1 - \frac{F^{\text{cache}}}{F^{\text{full}}}. \tag{29}$$

Table 5 reports FLOPs savings under different caching ratios ρ. For example, at $\rho = 0.7$, the measured FLOPs reduction is approximately $\mathbf{62.4}\%$ for OpenVLA and $\mathbf{56.1}\%$ for OpenVLA-OFT. Notably, OpenVLA-OFT exhibits larger absolute FLOPs, as its two-image input yields a longer prefill sequence and thereby exposes more backbone computations.

Table 5: FLOPs saving under different ρ.

Backend	ρ	FLOPs (T)	Saving
OpenVLA	0.0	1.86	–
	0.3	1.33	28.6%
	0.5	1.02	45.3%
	0.7	0.70	62.4%
OpenVLA-OFT	0.0	4.01	–
	0.3	3.03	24.5%
	0.5	2.38	40.7%
	0.7	1.76	56.1%

A.4 Experimental Details

Optimization Settings. All reported training runs are executed on a single node equipped with 8 NVIDIA H200 GPUs. We use AdamW optimization (Loshchilov & Hutter, 2017) and clip gradients

to a maximum norm of 1.0. Streaming training adopts grouped episode slots and two-frame truncated backpropagation through time (BPTT): the recurrent state is retained within each two-frame window and detached at window boundaries. For differentiable caching decisions with budget annealing, we linearly anneal the coefficient of the Gumbel noise (η_s in Eq. (12)) from 1.0 to 0, the temperature (τ_s in Eq. (12)) from 2.0 to 0.5, and the recomputation target from 0.7 (r_0 in Eq. (16)) to 0.2 ($r_\star$ in Eq. (16)) over $10{,}000$ steps. For linear memory warm-up, the coefficient for the temporal-change sensitivity (β in Eq. (17)) is fixed to 10, while the coefficients for the memory-conditioned gate correction (γ_s in Eq. (9)) and the consistency loss (λ_{cons} in Eq. (20)) are linearly scheduled over $10{,}000$ steps: the former rises from 0 to 1 and the latter decreases from 0.01 to 0.005.

OpenVLA. For the OpenVLA model with discrete actions, we employ action-token cross-entropy loss (Kim et al., 2024) with a per-process batch size of 4 (yielding an effective batch size of 32 under eight-way distributed training). Parameters associated with the memory-augmented caching decisions, including learnable queries, recurrent memory state, decision gate, and state-update module, are jointly trained. As OpenVLA adopts cross-entropy loss for discrete action tokens, gradients propagating to the caching gate traverse the vocabulary head and causal Transformer, resulting in weak gradient signals and challenging optimization. To address this issue, we enable LoRA with a rank of 64 and apply it exclusively to the last 8 decoder layers. This design provides an adaptable pathway to caching gate learning and preserve the pre-trained backbone. Specifically, we optimize for $30{,}000$ training steps, which constitutes less than 50% of the original OpenVLA fine-tuning schedule. The learning rate is set to 2×10^{-4} with $1{,}500$ warmup steps. Following linear learning-rate warmup, the learning rate remains constant for the rest of training. The coefficient of the computational budget loss (λ_{buget} in Eq. (20)) is set to 1.0 for this model.

OpenVLA-OFT. For OpenVLA-OFT with a continuous action head, we adopt the L1 regression loss, two input images (primary and wrist views), proprioceptive inputs, and image augmentation following the default configuration (Kim et al., 2025). The per-GPU batch size is 4, yielding an effective global batch size of 32 across eight GPUs. In this setup, LoRA is disabled: the pretrained VLA backbone and action head are frozen, and only parameters associated with our memory-augmented caching decisions are optimized. We train for $50{,}000$ steps, using a learning rate of 2×10^{-4} with a $1{,}500$-step warmup. After the linear warmup phase, OpenVLA-OFT employs cosine learning-rate decay, which decays the rate to 10% of its initial value by the end of training. The coefficient of the computational budget loss (λ_{buget} in Eq. (20)) is set to 0.1.

CogACT. For CogACT, which predicts continuous actions with a diffusion-based action model, we use the DiT-B action head and its noise-prediction mean-squared-error objective. We repeat the diffusion loss computation 8 times per batch and predict a future action window of 15 steps without using past action history following the default configuration (Li et al., 2024a). The pretrained VLM backbone and diffusion action head are frozen, and only parameters associated with our caching decisions are optimized. Training uses a global batch size of 128 (16 samples per GPU across eight NVIDIA H200 GPUs), BF16 mixed precision, and image augmentation. We train for $20{,}000$ optimizer steps with AdamW, a learning rate of 1×10^{-4}, and $2{,}000$ linear warmup steps, followed by cosine learning-rate decay. The coefficient of the computational budget loss λ_{buget} is set to 0.2.

A.5 Ablation on Memory Module Dimensions

We conduct extensive experiments to identify the optimal dimension of our learnable queries and memory state, with quantitative results summarized in Table 6. As we can see, for the OpenVLA model, the optimal setting is 128, which achieves the best task performance on the LIBERO *Spatial* suite. Specifically, a smaller dimension fails to provide sufficient representational capacity, thus restricting the module's learning ability and degrading performance. Conversely, excessively large dimensions introduce redundant learnable parameters, therefore complicating optimization and likewise impairing task accuracy. A similar trend holds for the OpenVLA-OFT model, where both overly small and overly large configurations yield suboptimal results. Nevertheless, OpenVLA-OFT exhibits considerably stable performance across different dimensions.

Table 6: Ablation studies on the dimension of our learnable queries and memory state using the LIBERO *Spatial* task.

Dimension	OpenVLA		OpenVLA-OFT	
	SR ↑	Ratio ↑	SR ↑	Ratio ↑
Base Model	84.4%	–	97.8%	–
+64	86.6%	0.70	99.0%	0.80
+128	**87.6%**	0.70	99.2%	0.80
+256	85.4%	0.70	**99.4%**	0.80
+512	85.8%	0.70	99.2%	0.80

Table 7: Ablation studies on components of our MemCache using the LIBERO benchmark.

Method	Success Rate ↑										Caching Ratio↑
	Spatial		Object		Goal		Long		Average		
OpenVLA	84.4%	–	86.6%	–	75.6%	–	53.2%	–	75.0%	–	–
+ Queries	85.4%	↑ 1.0%	87.5%	↑ 0.9%	75.6%	↑ 0.0%	56.8%	↑ 3.6%	76.3%	↑ 1.3%	0.65
+ Memory	86.2%	↑ 1.8%	86.4%	↓ 0.2%	77.0%	↑ 1.4%	56.2%	↑ 3.0%	76.5%	↑ 1.5%	0.65
+ Warmup	**87.6%**	**↑ 3.2%**	**87.6%**	**↑ 1.0%**	**78.2%**	**↑ 2.6%**	59.0%	**↑ 5.8%**	**78.1%**	**↑ 3.1%**	0.65
OpenVLA-OFT	97.8%	–	97.6%	–	97.6%	–	94.2%	–	96.8%	–	–
+ Queries	98.6%	↑ 0.8%	98.2%	↑ 0.6%	97.4%	↓ 0.2%	95.0%	↑ 0.8%	97.3%	↑ 0.5%	0.74
+ Memory	99.0%	↑ 1.2%	98.2%	↑ 0.6%	97.4%	↓ 0.2%	95.8%	↑ 1.6%	97.6%	↑ 0.8%	0.71
+ Warmup	**99.2%**	**↑ 1.4%**	**98.8%**	**↑ 1.2%**	**98.0%**	**↑ 0.4%**	**96.2%**	**↑ 2.0%**	**98.1%**	**↑ 1.3%**	**0.75**

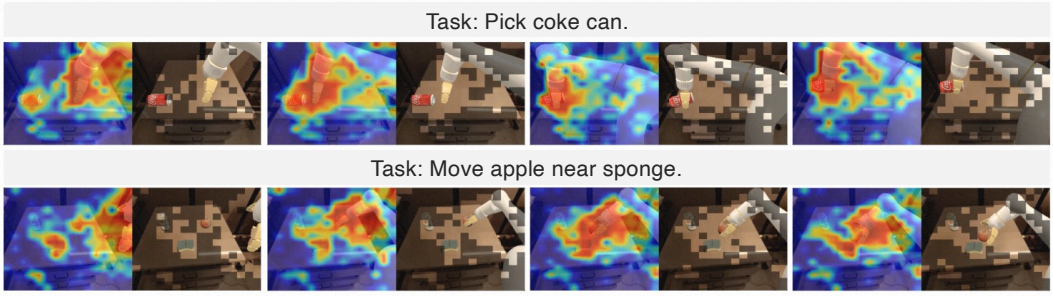

Figure 6: Visualization of our caching decisions for the *PickCan* and *MoveNear* tasks under the *Visual Matching* setting from the SIMPLER benchmark using CogACT. The left columns present heatmaps of our gate outputs: red denotes tokens selected for recomputation, and blue marks tokens chosen for caching. The right columns show the resulting caching behavior, with cached tokens masked to provide intuitive visual comparison.

For instance, the setting of 256 only outperforms 128 by a marginal 0.2% improvement in success rate. By comprehensively considering the results on both OpenVLA and OpenVLA-OFT, we ultimately adopt **128** as our default configuration.

A.6 Ablation on Memory Module Components

To validate the effectiveness of individual components of MemCache, we conduct ablation studies on the LIBERO benchmark and present the results in Table 7. Consistent with observations from Table 3, adopting the learnable-query-based caching decisions defined in Eq. (6) attains higher success rates compared with base models. This confirms the benefits of end-to-end optimization directly supervised by action signals, free from hand-crafted proxy signals. Incorporating our recurrent memory state from Eq. (9) yields additional $\uparrow 0.2\%$ and $\uparrow 0.3\%$ improvements in task performance. This illustrates its capability to refine caching decisions via history-driven corrections. Nevertheless, it also introduces extra learnable parameters, which raise optimization difficulty and demand sophisticated training strategies. For example, our memory-augmented variant even suffers a $\downarrow 1.1\%$ drop in task performance relative to the query-based method for the OpenVLA model evaluated on the LIBERO *Object* task. Motivated by this observation, we introduce the linear-warmup training schedule paired with our novel memory consistency loss in Sec. 3.4.2, which gradually activates memory-conditioned corrections to facilitate joint optimization. Under comparable caching ratios, this design delivers further gains in success rate of $\uparrow \mathbf{1.4}\%$ for OpenVLA and $\uparrow 0.2\%$ for OpenVLA-OFT, validating its effectiveness.

A.7 Qualitative Results

CogACT on SIMPLER. In addition to quantitative results on the SIMPLER benchmark for CogACT presented in Table 4, we provide qualitative results for better illustration, using the *PickCan* (grasp a single opened coke can in the scene) and *MoveNear* (move the source object near the target object) tasks under the *Visual Matching* setting as examples. As shown in Fig. 6, the focus of our gate consistently follows critical regions, including the robotic manipulator and task-relevant objects at every timestep, elucidating the mechanism behind the computation reduction achieved

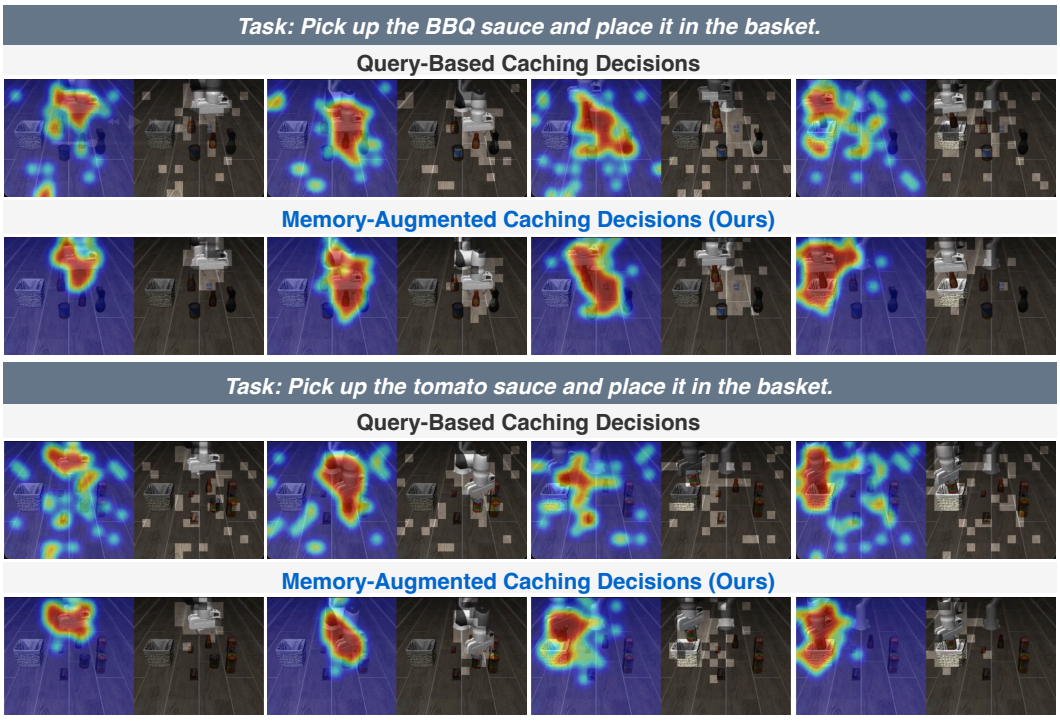

Figure 7: Comparison of query-based and our memory-augmented caching decisions on the LIBERO *Object* task using OpenVLA-OFT. The left columns present heatmaps of gates outputs, and the right columns show corresponding caching results. This comparison demonstrates that our gate exhibits more concentrated focus and thereby delivers more accurate caching decisions.

by our caching strategy, marked by only minor performance degradation and even improved task performance in some cases.

Comparison of Query-Based and Memory-Augmented Caching Decisions. To quantitatively compare query-based and our memory-augmented caching decisions, we visualize their gates outputs and corresponding caching results on the LIBERO Object task under the OpenVLA-OFT setting. As observed in Fig. 7, the focus of our gate is more concentrated on critical regions, including the robotic arm and salient objects. By contrast, the query-based gate tends to be distracted by noise from static backgrounds and irrelevant objects. This verifies the effectiveness of our proposed memory-conditioned corrections in mitigating gate oscillations caused by transient visual noise and yielding more reliable caching decisions.

Visualizations on All Four LIBERO Task Suites. We supplement our analysis with additional visualizations of MemCache's token caching results across all four LIBERO task suites under the OpenVLA-OFT setting, aiming to provide clearer and more comprehensive insights into our caching mechanism. These qualitative results cover four distinct scenarios: recognizing varied spatial locations, identifying unseen objects, obeying task instructions, and handling long-horizon tasks. Specifically, as shown in the first two rows of Fig. 8, which offers qualitative results for the *Spatial* suite evaluating the model's generalization across various object spatial relations, our gate correctly captures objects at diverse spatial locations, confirming its capacity to reason over spatial positions. The third and fourth rows present results from the *Object* suite, which validates generalization toward unseen object instances. Our gate reliably locates target objects while disregarding irrelevant ones, demonstrating its ability to pick out salient information and filter unimportant visual content. The fifth and sixth rows correspond to the *Goal* suite with diverse task outcomes. Here, the gate faithfully follows task instructions and exhibits effective instruction compliance under end-to-end optimization. Lastly, the final two rows visualize outputs from the *Long* suite for long-horizon tasks. The gate consistently tracks frame-wise critical regions, such as the robotic arm and salient objects, and suppresses unimportant background areas. This observation verifies that our caching gate maintains focus on task-relevant visual cues over extended sequential manipulation.

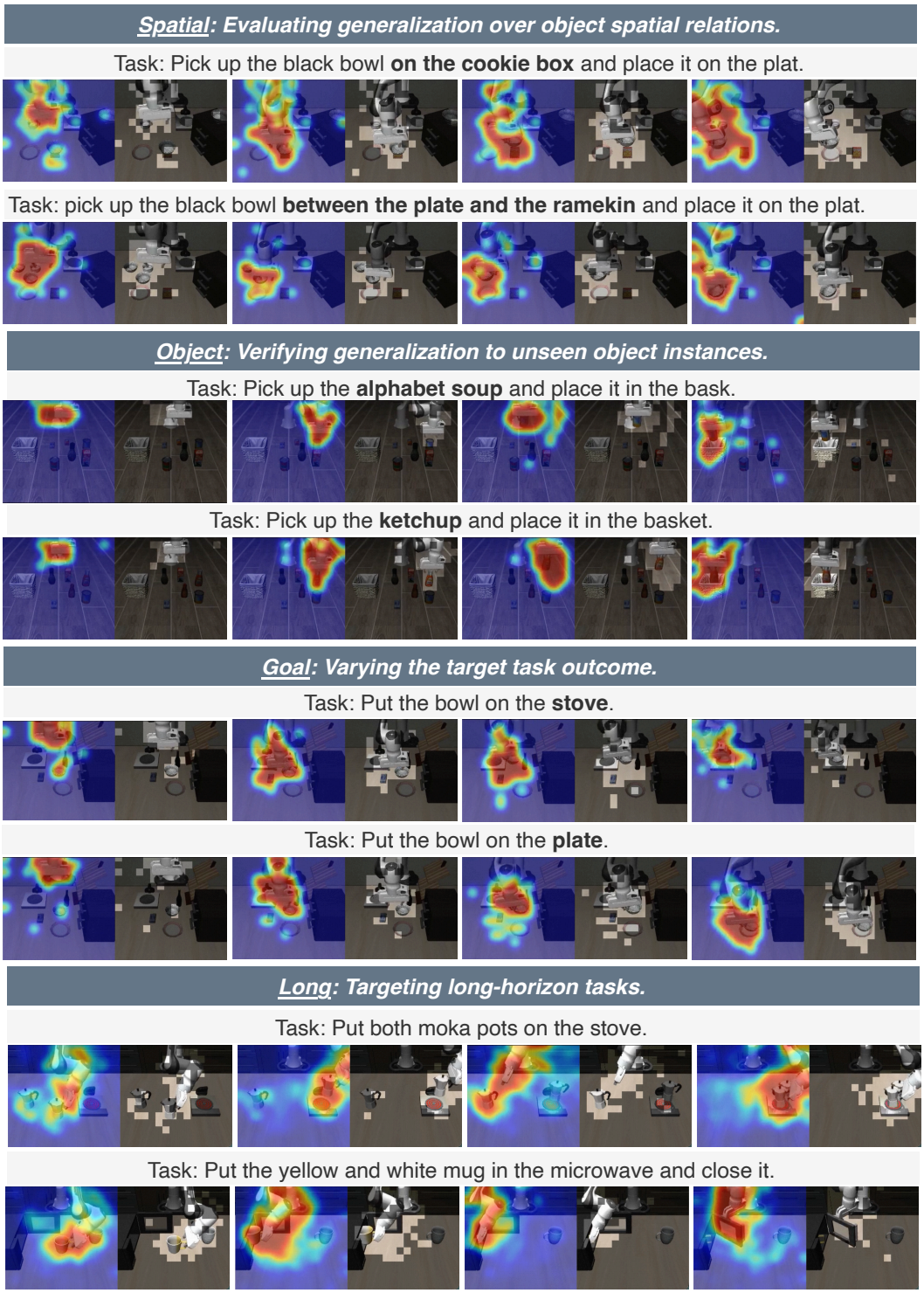

Figure 8: Visualization of our caching decisions across *all four task suites* from the LIBERO benchmark under the OpenVLA-OFT setting. The left columns present heatmaps of our gate outputs, and the right columns show the resulting caching behavior. These qualitative results validate our effectiveness in four distinct scenarios: recognizing varied spatial locations, identifying unseen objects, following task instructions, and handling long-horizon tasks.