LPA-CACHE: LATENCY-PLATEAU AWARE KV CACHE REUSE FOR EFFICIENT VISION-LANGUAGE-ACTION MODEL INFERENCE ON GPUS

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-language-action (VLA) models require low-latency inference for responsive robotic control. Previous works have shown that cross-frame reuse of visual key-value (KV) representations can reduce redundant computation. However, reductions in FLOPs do not necessarily translate into proportional latency reductions. Profiling the prefill-stage of VLA language-backbone on GPUs reveals a pronounced latency-plateau effect in dominant linear operators: reducing the number of recomputed tokens within a plateau yields marginal speedup, whereas crossing a plateau boundary yields substantial latency reduction. Motivated by this observation, we propose LPA-CACHE, a training-free method for accelerating VLA inference on GPUs through latency-plateau-aware KV cache reuse. A one-time profiling procedure identifies latency-plateau boundaries on the target GPU platform. These boundaries are combined with preselected reuse layers to construct a fixed-shaped reuse schedule. At inference time, visual tokens estimated to be task-relevant from previous-step attentions are protected from reuse, while the remaining visual tokens are ranked by inter-frame relative changes and progressively reused according to layerwise reuse budgets. Guided by observed cross-layer consistency in attention-based task relevance, LPA-CACHE extracts attention weights from only a small subset of layers, allowing FlashAttention-2 to be used in the remaining layers for further speedup. Extensive experiments across several representative VLA models on two robotic simulation platforms (LIBERO and CALVIN) demonstrate that LPA-CACHE achieves up to 1.38× language-backbone prefill speedup and up to 1.15× observation-to-action inference speedup with negligible loss in task success rate.

1 INTRODUCTION

Vision-language-action (VLA) models map visual observations and language instructions to robot actions, offering a unified approach to language-conditioned manipulation (Brohan et al., 2023; Kim et al., 2024; Black et al., 2026). Low inference latency is critical for closed-loop deployment, where robots must respond promptly to environment changes. Language-backbone prefill, i.e. the processing of visual and language input before action generation, remains a substantial latency contributor across several representative VLA models.

Previous work has shown that temporal redundancy in visual observations can be exploited to reduce prefill computation by reusing cached key-value (KV) representations at visual-token positions with small changes between frames (Xu et al., 2025). Unlike token pruning, cross-frame KV reuse skips current-frame recomputation at selected positions while retaining their cached keys and values for attention.

However, reductions in computation do not necessarily translate into proportional inference latency reductions on GPUs. Our latency breakdown shows that linear operators dominate the language-backbone prefill latency in most of the VLA models we study. Profiling these operators reveals a pronounced latency-plateau effect. For a fixed operator and execution configuration, latency increases in a staircase-like pattern with the number of input tokens: changes within a plateau have limited impact, whereas crossing a plateau boundary produces a pronounced latency jump. This behavior

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.

persists in decoder-layer measurements using the actual KV-reuse execution path. Consequently, the same reduction in recomputed tokens can yield markedly different latency savings depending on whether it crosses a plateau boundary. This inspires choosing reuse budgets that reach lower-latency plateaus without unnecessarily reducing the number of recomputed tokens further within them.

Motivated by this observation, we propose LPA-CACHE, a training-free, latency-plateau-aware method for cross-frame KV cache reuse. A one-time offline profiling step identifies useful latency-plateau boundaries for the target model on the deployment GPU platform. We then construct a fixed-shaped reuse schedule, whose layerwise reuse budgets are aligned with the profiled latency boundaries. At inference time, visual tokens estimated to be task-relevant from previous-step attentions are explicitly protected from reuse and recomputed. The remaining visual tokens are ranked by their relative inter-frame changes, with smaller-changed tokens selected for reuse at earlier stages. Starting at their assigned reuse-entry layers, selected positions are removed from the active hidden-state stream, the corresponding KV entries are carried over from the cache available after the previous inference step rather than updated, and remain accessible to active queries under the model's attention mask.

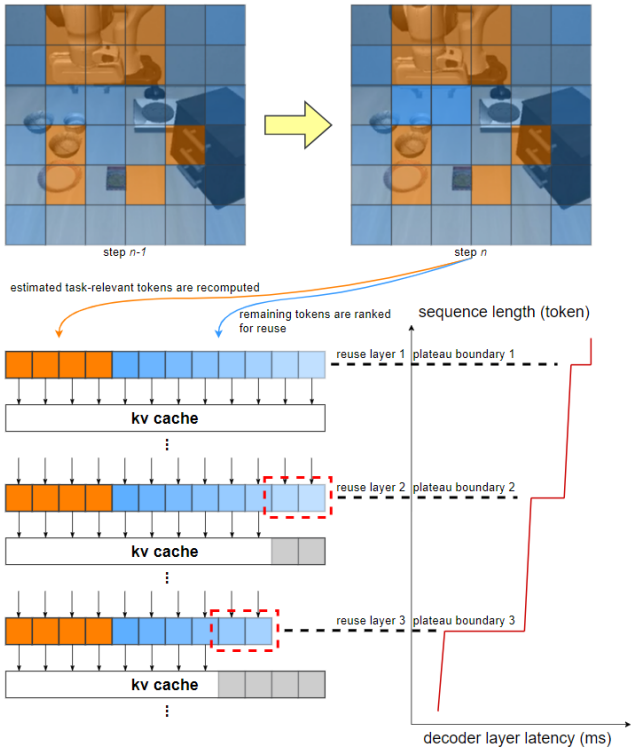


Figure 1: Conceptual overview of LPA-CACHE.

Our attention analysis also reveals the *cross layer consistency* in attention-based visual token task relevance: visual positions receiving high text-to-vision attentions often recur across layers. This observation makes it practical to extract attention weights from only a small subset of representative layers for task relevance estimation, allowing FlashAttention-2 (Dao et al., 2022; Dao, 2023) to be used in all remaining layers.

The main contributions are threefold.

1. We identify the mechanism behind efficient KV cache reuse for VLA inference on GPUs. Specifically, we show that inference latency exhibits a latency-plateau effect, and that the benefit of token reuse depends on whether the active sequence length crosses plateau boundaries rather than only on the amount of reduced computation.

2. We propose LPA-CACHE, a training-free and hardware-aware KV cache reuse framework for VLA inference. LPA-CACHE combines latency-plateau-aware schedule construction, attention-based protection of task-relevant tokens, and change-based reuse prioritization, while remaining compatible with efficient attention implementations.

3. We validate LPA-CACHE on representative VLA models and robotic manipulation benchmarks. Experiments demonstrate that LPA-CACHE consistently improves inference efficiency while maintaining competitive task performance, showing the effectiveness of hardware-aware reuse budgeting for practical VLA acceleration.

2 Related Work

As vision-language-action (VLA) models are increasingly applied to closed-loop robotic control, improving their inference efficiency has become an important research problem. Existing methods can be broadly grouped into four categories: efficient action generation, visual token reduction and quantization, temporal KV cache reuse, and hardware-aware inference.

2.1 Efficient Inference for VLA Models

Recent work has improved VLA inference efficiency from different perspectives. Some methods focus on the action generation stage, such as parallel action prediction, action chunking, and compact action tokenization (Kim et al., 2025; Song et al., 2026; Pertsch et al., 2025). Others target the visual-language backbone, where long multimodal sequences and repeated per-layer computation often dominate the inference cost. Our work belongs to the latter line and specifically focuses on accelerating the language-backbone prefill stage.

2.2 Visual Token Reduction, Pruning, and Quantization

Reducing the computation applied to visual tokens is a common strategy for efficient multimodal inference. Prior studies explore token pruning and sparsification in vision-language models to remove redundant visual tokens while preserving important information (Chen et al., 2024; Zhang et al., 2025), and similar ideas have been extended to VLA models with action-aware criteria (Liu et al., 2026; Pei et al., 2025). Quantization is another complementary direction, which reduces memory and computation cost through lower numerical precision (Xu et al., 2026; Zhang et al., 2026).

2.3 Temporal KV Cache Reuse for VLA Inference

Another line of work exploits temporal redundancy between consecutive observations. VLA-CACHE (Xu et al., 2025) shows that reusing cached visual KV representations can effectively reduce VLA inference cost. Follow-up studies extend this idea with different reuse criteria and control mechanisms, such as static-dynamic token disentanglement (Qiu et al., 2026), confidence-gated recomputation (Wu et al., 2026), action-tolerance-based stale-KV reuse or pruning (Li et al., 2026), and learned adaptive caching policies (Wei et al., 2026). In contrast to these methods, our work emphasizes latency-oriented reuse budgeting rather than computation reduction alone.

2.4 Hardware-Aware Inference

Recent studies have shown that lower FLOPs do not necessarily lead to proportional runtime reduction on modern GPUs. Prior work attributes this behavior to hardware factors such as tiled execution, thread-block scheduling, and wave quantization (Anthony et al., 2024; NVIDIA, 2023). These observations suggest that effective acceleration should consider actual hardware latency characteristics. Motivated by this insight, our work explicitly incorporates GPU latency plateaus into the design of KV cache reuse for VLA inference.

3 Methodology

The architecture of LPA-CACHE is demonstrated by Fig 2.

In this section, we first characterize the latency breakdowns of the representative VLA models we study and demonstrate how prefill latency changes with the number of recomputed tokens. Then, we present LPA-CACHE's reuse strategy and explain the rationale behind reuse-candidate selection and prioritization.

3.1 Inference Latency Breakdown and the Latency-Plateau Effect

Inference latency breakdown. In the representative VLA models we study, prefill-stage linear operators contribute substantially to observation-to-action inference latency. The clustered column diagram in Figure 3 gives a hierarchical latency breakdown. As shown, prefill stage accounts for

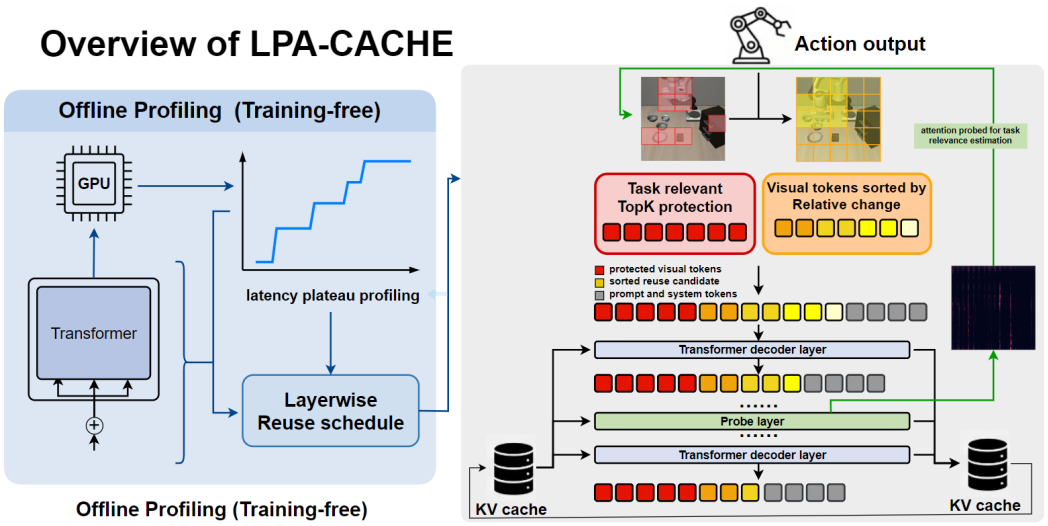


Figure 2: Architecture of LPA-CACHE: LPA-CACHE consists of an offline profiling stage and an online inference stage. Offline profiling identifies latency-plateau boundaries on the target GPU platform and constructs a fixed-shape reuse schedule. During online inference, task-relevant visual tokens are protected using attention-based relevance estimation from a small number of probe layers, while the remaining tokens are ranked by relative inter-frame embedding changes and reused according to the schedule.

39.26%, 87.76%, and 55.56% of the total inference latency in the three models, respectively. In prefill-stage, language-backbone linear operators, including attention projections and feed-forward projections, account for 27.69%, 52.19%, and 35.89% of the overall end-to-end inference latency. These results identify linear operators as an important acceleration target for reducing overall inference latency.

The latency-plateau effect. We profile three typical linear operators in OpenVLA-OFT on a NVIDIA RTX 3090 Ti as a typical example, using batch size 1 and varying the active input sequence length m. The profiled operators include $d \to d$ query projection, $d \to d_{\text{ff}}$ up/gate projection, and $d_{\text{ff}} \to d$ down projection, where $d = 4096$ and $d_{\text{ff}} = 11008$.

As shown in Figure 4, the latencies of the three linear operators exhibit a pronounced staircase-like pattern as the active sequence length m, i.e., the number of recomputed tokens, increases. Within a plateau, latency remains nearly constant despite changes in m, whereas pronounced jumps occur at particular sequence-length thresholds. Consequently, reducing the number of recomputed tokens within a plateau yields little reduction in linear-operator latency, while even a small token reduction that crosses into a lower-latency plateau can produce substantial latency savings. This behavior is consistent with the tiled execution and wave-quantization effects discussed in prior work (Anthony et al., 2024; NVIDIA, 2023).

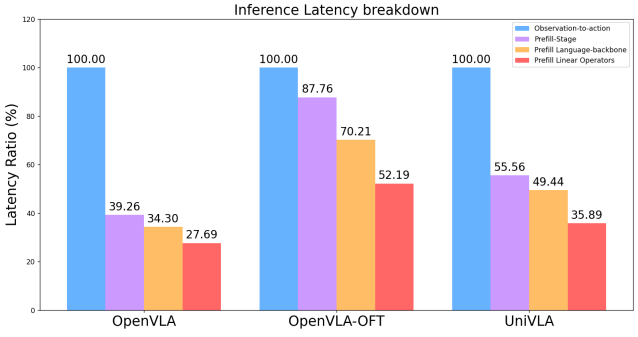


Figure 3: Inference latency breakdown of OpenVLA, OpenVLA-OFT and UniVLA.

Decoder-layer profiling under KV reuse. To identify the impact of the linear latency-plateau on the actual execution path, we also profile a complete decoder layer. The active hidden-state sequence has length m, while the KV sequence retains its full length N. Active positions update their KV entries, while reused positions retain their cached entries. Thus, self-attention operates with m queries and N keys and values, rather than shortening both the query and the KV as in token pruning.

For a complete decoder layer, the latency curve continues to exhibit pronounced stairs, with its major boundaries closely matching those of the dominant linear operators. These measurements show that the operator-level *latency-plateau effect* persists at the decoder-layer level, making it directly relevant to the language-backbone latency. Unlike the nearly flat plateaus of isolated linear operators, decoder-layer latency increases gradually within plateaus as the active sequence length grows, since a decoder-layer also includes operations such as RMSNorm, attention matrix multiplications, and softmax, whose sequence length-dependent costs contribute to this gradual increase.

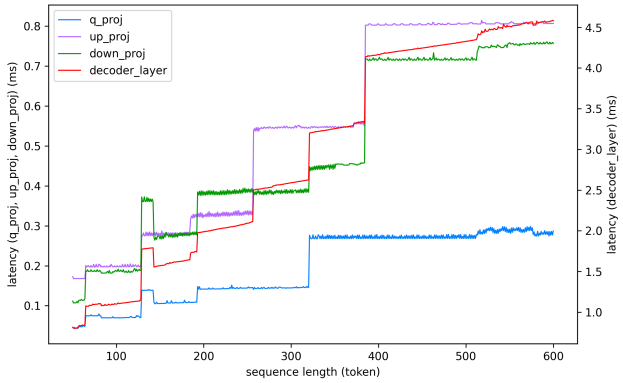


Figure 4: The profiled latency-plateau effect of OpenVLA prefill linear operators

Implication for reuse budgeting. For a fixed execution configuration, let $T_\ell(m; N)$ denote the measured latency of layer ℓ with m active positions and a full-length KV sequence of size N. Reusing r additional positions changes its latency by

$$\Delta T_\ell(m, r; N) = T_\ell(m; N) - T_\ell(m - r; N). \tag{1}$$

The same r can produce markedly different savings depending on the starting length m and the boundaries crossed. Inspired by this observation, LPA-CACHE aligns the remaining active length m_ℓ at reuse-entry layer ℓ to the closest latency-plateau boundary. The underlying principle is to obtain as much latency reduction as possible with as few additional reused positions as possible, thereby achieving a better speedup-success-rate trade-off. Strictly formulated prefill-stage language-backbone latency is delivered in Appendix B.

Table 1 shows an ablation experiment on OpenVLA using the LIBERO Spatial subtask. We choose layer 6 as reuse entry layer, and vary the remaining active length m. It can be seen that when m crosses two of the profiled latency-plateau boundaries, i.e. $m = 256$ and $m = 192$, reusing $r = 1$ additional position brings marked latency saving with hardly any degradation on task success rate. However, when m varies within a plateau, shortened active sequence lowers task performance, but brings only marginal latency reduction.

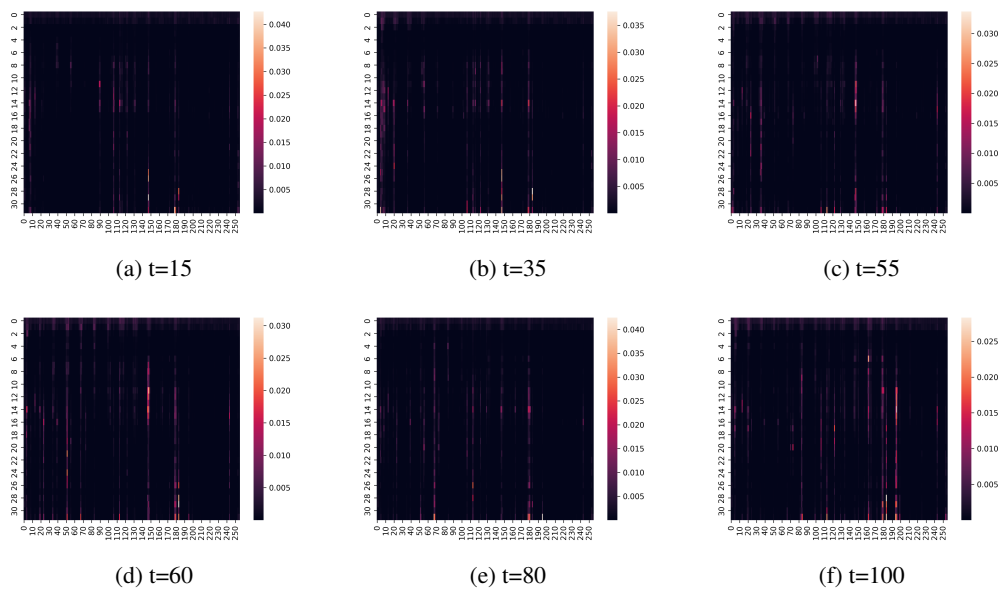
m	latency	success rate
291	77.68 ms	79.1%
257	75.23 ms	77.3%
256	**69.05 ms**	**77.3%**
255	68.87 ms	77.3%
193	64.91 ms	76.2%
192	**58.34 ms**	**76.1%**
191	58.01 ms	76.1%

Table 1: Ablation experiment on remaining active sequence length m

3.2 Reuse Candidate Selection and Prioritization

Given the reuse budget, visual tokens are adaptively selected and prioritized for reuse. We adopt two complementary methods: visual tokens estimated to be task-relevant from previous-step attentions are protected from reuse, while the remaining tokens are prioritized for reuse according to their relative inter-frame embedding changes.

Attention-based protection. Let $\mathcal{V}$ and $\mathcal{P}$ denote the visual token positions and the language prompt positions, respectively. Let $A_{t,h}^{(\ell)}[q, i]$ be the post-softmax attention weight from query posi-



(a) t=15 (b) t=35 (c) t=55

(d) t=60 (e) t=80 (f) t=100

Figure 5: Heatmap of $a_{t,i}^{(\ell)}$ tracked from the same UniVLA simulation episode on LIBERO benckmark, with timestep $t = 15, 35, 55, 60, 80, 100$ respectively.

tion q to key position i in head h of layer ℓ at inference step t. We define the *layerwise visual token task relevance* of position i as

$$a_{t,i}^{(\ell)} = \frac{1}{H_\ell |\mathcal{P}|} \sum_{h=1}^{H_\ell} \sum_{q \in \mathcal{P}} A_{t,h}^{(\ell)}[q, i], \qquad i \in \mathcal{V}. \tag{2}$$

It should be noted that $a_{t,i}^{(\ell)}$ is an attention-based proxy rather than a direct measure of a token's contribution to the task success rate.

Let $\mathcal{S}$ be a small subset of *task relevance probe layers*, we average their task relevance estimates and protect the n highest-scoring visual positions from reuse:

$$\bar{a}_{t-1,i} = \frac{1}{|\mathcal{S}|} \sum_{\ell \in \mathcal{S}} a_{t-1,i}^{(\ell)}, \qquad \mathcal{I}_{t-1} = \mathrm{TopK}_{i \in \mathcal{V}}\big(\bar{a}_{t-1,i}, n\big). \tag{3}$$

Attention weights from the previous step are used because current-step decoder attention weights are not yet available when reuse candidates are selected.

Relative embedding change. Let $\mathbf{e}_{t,i} \in \mathbb{R}^d$ denote the visual embedding at position i entering the language backbone, its relative inter-frame change is defined as

$$\delta_{t,i} = \frac{\|\mathbf{e}_{t,i} - \mathbf{e}_{t-1,i}\|_2}{\max\big(\|\mathbf{e}_{t-1,i}\|_2, \epsilon\big)}, \tag{4}$$

where $\epsilon > 0$ ensures numerical stability. We then construct the candidate set and its ordered reuse priority:

$$\mathcal{C}_t = \mathcal{V} \setminus \mathcal{I}_{t-1}, \qquad \boldsymbol{\pi}_t = \mathrm{argsort}^{\uparrow}_{i \in \mathcal{C}_t} \delta_{t,i}. \tag{5}$$

Positions with smaller relative embedding changes are placed earlier in $\boldsymbol{\pi}_t$ and prioritized for assignment to earlier reuse-entry layers, while protected positions $\mathcal{I}_{t-1}$ remain active. Unlike cosine similarity, relative change responds to both directional and magnitude changes.

We compute the relative change using visual embeddings already available in the inference pipeline. These embeddings retain spatially indexed visual information while incorporating contextual information from the vision backbone, and are directly adjacent to the language backbone where reuse takes place.

We heuristically explain the effectiveness of this selection strategy as follows: Previous-step attentions identifies positions that were potentially task-relevant, but robot or object motion can cause task-relevant content to move into previously task-irrelevant positions. A large relative inter-frame embedding change may indicate that task-relevant content has entered the corresponding spatial region in the current frame. Assigning a lower reuse priority to such positions therefore provides a safeguard against the temporal lag in attention-based relevance estimates.

Inter-layer consistency of prefill attention Figure 5 visualizes $a_{t,i}^{(\ell)}$ across all decoder layers and visual positions. The recurring highlighted vertical bands in the heatmap indicate that visual positions receiving high attention in one layer tend to remain salient in other layers, despite the differences in attention magnitude and some layer-specific patterns. This observed *cross-layer consistency* persistently exists during the whole execution episode, making it practical to extract attention weights from only a small set of probe layers for task relevance estimation, rather than extracting attention weights from all layers. This allows FlashAttention-2(Dao, 2023) to be applied to the remaining layers for further speed up.

4 IMPLEMENTATION

Action-generation interfaces. OpenVLA (Kim et al., 2024) autoregressively generates discretized action tokens, whereas the continuous-regression configuration of OpenVLA-OFT (Kim et al., 2025) predicts an action chunk in parallel from appended action-slot hidden states using an L1-trained action head.

LPA-CACHE removes only visual positions from the active hidden-state sequence while retaining full-length layerwise KV caches. For OpenVLA, the KV context required for auto-regressive action decoding are retained. For OpenVLA-OFT, all action slots and the terminal stop token remain active, action-slot hidden states can be extracted from the suffix of the final hidden states with correct number and ordering despite the shortened active sequence. Neither model therefore requires additional model-specific adjustment.

UniVLA adaptation. UniVLA (Bu et al., 2025) autoregressively predicts latent-action tokens and uses a separate action decoder to produce robot actions. The shortened final hidden states cannot be directly applied to its action decoder: doing so can omit required visual features or shift non-visual states into the selected readout range, causing severe performance degradation. We therefore introduce an additional final-hidden-state cache for to tackle this mismatch.

Let $\mathbf{H}_t \in \mathbb{R}^{N\times d}$ denote the full-length final hidden states produced by the language backbone during a full prefill, i.e., the normalized output after the last decoder layer, where the batch dimension is omitted. Under LPA-CACHE, only $m = m_{L-1}$ positions are recomputed, yielding compact final hidden states $\widetilde{\mathbf{H}}_t \in \mathbb{R}^{m\times d}$. Let $\mathbf{p}_t = (p_{t,1}, \ldots, p_{t,m})$ map its rows to their original sequence positions, and let $\mathcal{A}_t = \{p_{t,1}, \ldots, p_{t,m}\}$. We maintain a full-length final-hidden-state cache $\mathbf{C}_t \in \mathbb{R}^{N\times d}$, updated as

$$\mathbf{C}_t[i,:] = \begin{cases} \widetilde{\mathbf{H}}_t[k,:], & i = p_{t,k}, \quad k = 1, \ldots, m, \\ \mathbf{C}_{t-1}[i,:], & i \notin \mathcal{A}_t. \end{cases} \tag{6}$$

The action decoder then receives $\widehat{\mathbf{E}}_t^{\mathrm{vis}} = \mathbf{C}_t[\boldsymbol{\nu},:]$, where $\boldsymbol{\nu}$ preserves the original decoder's visual-feature readout shape and ordering. This adaptation preserves the action decoder's input structure, requiring only one $N \times d$ buffer and indexed writes without restoring full-length computation inside the decoder layers.

The execution procedure of the LPA-CACHE's inference algorithm is defined in Appendix C.

5 EXPERIMENT

5.1 EXPERIMENTAL SETUP

Platform and models. To validate the effectiveness of LPA-CACHE, we evaluate our method on three open-source VLA models: OpenVLA(Kim et al., 2024), OpenVLA-OFT(Kim et al., 2025)

Table 2: Comparison of different VLA acceleration methods on the LIBERO benchmark.

Method	Success Rate(%)					Latency(ms)	
	Spatial	Object	Goal	Long	Average	Prefill Stage	Observation-to-Action
OpenVLA	79.1	72.7	76.5	56.0	71.1	79.03	232.58
+ FlashAttn-2	78.6	72.4	76.7	55.3	70.8	77.68	228.87
+ VLA-CACHE	75.2	69.3	75.5	53.8	68.5	66.50	219.52
+ LPA-CACHE	**76.1**	69.3	**78.0**	**54.2**	**69.4**	**58.34**	**208.61**
OpenVLA-OFT	98.6	98.5	97.4	93.1	96.9	150.29	188.72
+ FlashAttn-2	98.4	98.3	98.0	92.9	96.9	134.60	171.17
+ VLA-CACHE	97.2	97.7	97.3	90.6	95.7	142.92	184.53
+ LPA-CACHE	**97.3**	**98.4**	**97.8**	**91.5**	**96.3**	**114.27**	**151.26**
UniVLA	96.6	95.2	96.4	93.1	95.3	81.01	162.60
+ FlashAttn-2	96.5	95.3	96.2	92.9	95.2	77.62	159.83
+ VLA-CACHE	96.0	94.8	95.7	92.1	94.7	65.47	151.28
+ LPA-CACHE	**96.3**	**94.9**	**96.1**	**92.4**	**94.9**	**56.93**	**139.62**

Table 3: Comparison of different VLA acceleration methods on the CALVIN benchmark.

Method	Average	Success Rate					Latency(ms)	
	Length	SR1	SR2	SR3	SR4	SR5	Prefill Stage	Observation-to-Action
UniVLA	3.89	94.6	87.0	77.4	70.2	59.8	79.15	159.42
+ FlashAttn-2	3.89	94.8	86.8	77.0	70.2	60.2	77.93	157.61
+ VLA-CACHE	3.748	94.4	86.2	72.6	66.4	55.2	64.28	148.67
+ LPA-CACHE	**3.816**	**95.8**	**86.2**	**75.6**	**66.8**	**57.2**	**56.62**	**137.48**

and UniVLA(Bu et al., 2025), using the LIBERO(Liu et al., 2023) benchmark and CALVIN(Mees et al., 2022) benchmark. All experiments are conducted on a single NVIDIA RTX 3090 Ti GPU with 24 GB memory.

Compared Baseline. We adopt VLA-CACHE (Xu et al., 2025), a representative temporal KV cache reuse method, as a compared baseline. For OpenVLA and OpenVLA-OFT, we deploy the official open-source implementation of VLA-CACHE. For UniVLA, we port VLA-CACHE using the action-decoder interface adjustment described in Sec. 4, so that the baseline can operate correctly on UniVLA.

Metrics. We report three metrics. *Task success rate* measures the specified task performance in simulation. *Language-backbone prefill latency* measures the CUDA runtime of the language backbone in the prefill stage, which is directly accelerated by LPA-CACHE. *Vision-to-action latency* measures the wall-clock latency from the input visual observation to the emitted action. It covers the full observation-to-action pipeline, including the additional overhead introduced by the reuse algorithm, such as reuse decision making and cache maintenance.

5.2 BENCHMARKS

LIBERO. LIBERO (Liu et al., 2023) is a simulation benchmark for language-conditioned robot manipulation. It contains four task suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-Long. These suites evaluate spatial reasoning, object-centric manipulation, goal-conditioned behavior, and long-horizon multi-step execution, respectively.

CALVIN. CALVIN (Mees et al., 2022) is a simulated benchmark for long-horizon, language-conditioned manipulation. Its continuous task evaluation places the policy in a scene and presents a sequence of language instructions that must be executed consecutively without resetting the environment. In this work, we use the CALVIN simulator for closed-loop continuous task evaluation.

5.3 Simulation Experiment Results

Schedule settings. Following the schedule formulation in Appendix B, we adopt a single reuse entry ($J = 1$) for the main experiments. For OpenVLA and UniVLA, we empirically choose $(b_1, M_1) = (6, 192)$, with task-relevance probe layers $\{15\}$ for OpenVLA and $\{14, 15\}$ for UniVLA. For OpenVLA-OFT, we use $(b_1, M_1) = (11, 384)$ and probe layer $\{15\}$.

FlashAttention-2 compatibility. Since the original baselines of all three models are FlashAttention-2 compatible, a fair latency comparison should take FlashAttention-2 baselines into account. VLA-CACHE requires attention weights from all layers to compute its layerwise reuse schedule, and therefore cannot use FlashAttention-2. This loss can partially offset its reuse gains. In contrast, LPA-CACHE extracts attention only at the probe layers and uses FlashAttention-2 in all remaining layers.

LIBERO main results. Table 2 reports the LIBERO results. For OpenVLA, relative to the FlashAttention-2 baseline, LPA-CACHE reduces language-backbone prefill latency from 77.68 ms to 58.34 ms ($1.33\times$) and observation-to-action latency from 228.87 ms to 208.61 ms ($1.10\times$). Its average success rate is 69.4%, higher than VLA-CACHE (68.5%) and within 1.4 points of the FlashAttention-2 baseline (70.8%). For UniVLA, LPA-CACHE reduces prefill latency from 77.62 ms to 56.93 ms ($1.36\times$) and observation-to-action latency from 159.83 ms to 139.62 ms ($1.14\times$), while achieving 94.9% average success, compared with 94.7% for VLA-CACHE and 95.2% for the FlashAttention-2 baseline. For OpenVLA-OFT, LPA-CACHE reduces prefill latency from 134.60 ms to 114.27 ms ($1.18\times$) and observation-to-action latency from 171.17 ms to 151.26 ms ($1.13\times$), with 96.3% average success versus 95.7% for VLA-CACHE and 96.9% for the FlashAttention-2 baseline. The benefit of FlashAttention-2 compatibility is most pronounced on OpenVLA-OFT: VLA-CACHE incurs 142.92 ms prefill and 184.53 ms observation-to-action latency, even exceeding the FlashAttention-2 baseline (134.60 ms and 171.17 ms), whereas LPA-CACHE remains substantially faster. This is because OpenVLA-OFT uses an additional dynamic wrist-camera observation, leading to a longer prefill sequence on which FlashAttention-2 has a larger effect. For OpenVLA and UniVLA, which use a single fixed third-person view, the FlashAttention-2 gain is smaller (about 2–3 ms), VLA-CACHE thus still obtains moderate acceleration, but LPA-CACHE consistently achieves lower latency.

CALVIN results. Table 3 shows the CALVIN results for UniVLA. LPA-CACHE achieves an average task length of 3.816, compared with 3.748 for VLA-CACHE and 3.89 for the FlashAttention-2 baseline. It also improves SR1 from 94.4% to 95.8% and SR5 from 55.2% to 57.2% over VLA-CACHE. In terms of latency, relative to the FlashAttention-2 baseline, LPA-CACHE reduces prefill latency from 77.93 ms to 56.62 ms ($1.38\times$) and observation-to-action latency from 157.61 ms to 137.48 ms ($1.15\times$). These results are consistent with the LIBERO findings. The detailed subtask success rates is provided in Appendix D.

6 Conclusion

We presented LPA-CACHE, a training-free, latency-plateau-aware KV cache reuse method for VLA inference. LPA-CACHE reduces the active hidden-state sequence at a small number of reuse-entry layers while retaining full-length KV caches, and protects task-relevant visual tokens from reuse. By aligning layerwise reuse budgets to the profiled latency-plateau boundaries, it converts a small amount of token reuse into a substantial latency reduction. Experiments on OpenVLA, OpenVLA-OFT, and UniVLA across LIBERO and CALVIN show up to $1.38\times$ prefill-stage language-backbone speedup and up to $1.18\times$ observation-to-action speedup with negligible loss in task success rate. These results demonstrate that hardware-aware KV reuse budgeting can effectively improve speedup–success-rate trade-off.

AI USE STATEMENT

We used generative AI tools to assist manuscript preparation, literature search and coding. For the manuscript, AI tools assisted with translating, refining wording, and checking grammar. For literature search, AI tools assisted with searching potentially relevant publications. For coding, AI tools assisted with source code understanding and a small amount of programming, and were not used to implement the core method.

The authors organized the literature and manually incorporated citations and bibliography entries into the manuscript; AI tools were not used to generate the manuscript's citations or reference list. The authors manually verified that all cited references exist and correspond to the cited works. The authors reviewed and revised all AI-assisted text. All AI-assisted code was tested and verified by the authors.

We take responsibility for the final content of this work, including its text, references, claims, and conclusions.

REFERENCES

Quentin Anthony, Jacob Hatef, Deepak Narayanan, Stella Biderman, Stas Bekman, Junqi Yin, Aamir Shafi, Hari Subramoni, and Dhabaleswar Panda. The case for co-designing model architectures with hardware, 2024. URL https://arxiv.org/abs/2401.14489.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2026. URL https://arxiv.org/abs/2410.24164.

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

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Univla: Learning to act anywhere with task-centric latent actions, 2025. URL https://arxiv.org/abs/2505.06111.

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, 2024. URL https://arxiv.org/abs/2403.06764.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL https://arxiv.org/abs/2307.08691.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

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

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. URL https://arxiv.org/abs/2502.19645.

Zhuoyuan Li, Rui Zhao, Jin Wang, Hanwei Zhu, Cong Zhang, Giuseppe Valenzise, Weisi Lin, and Kin-Man Lam. Just noticeable difference modeling for token compression in vision-language-action models, 2026. URL https://arxiv.org/abs/2608.21247.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning, 2023. URL https://arxiv.org/abs/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, 2026. URL https://arxiv.org/abs/2511.16449.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks, 2022. URL https://arxiv.org/abs/2112.03227.

NVIDIA. *Matrix Multiplication Background User's Guide*, February 2023. URL https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html. Document DU-09799-001_v001.

Xiaohuan Pei, Yuxing Chen, Siyu Xu, Yunke Wang, Yuheng Shi, and Chang Xu. Action-aware dynamic pruning for efficient vision-language-action manipulation, 2025. URL https://arxiv.org/abs/2509.22093.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models, 2025. URL https://arxiv.org/abs/2501.09747.

Weikang Qiu, Huashuo Lei, Tinglin Huang, and Rex Ying. Efficient long-horizon vision-language-action models via static-dynamic disentanglement, 2026. URL https://arxiv.org/abs/2602.03983.

Wenxuan Song, Jiayi Chen, Pengxiang Ding, Han Zhao, Wei Zhao, Zhide Zhong, Zongyuan Ge, Zhijun Li, Donglin Wang, Jun Ma, Lujia Wang, and Haoang Li. Pd-vla: Accelerating vision-language-action model integrated with action chunking via parallel decoding, 2026. URL https://arxiv.org/abs/2503.02310.

Yujie Wei, Jiahan Fan, Jiyu Guo, Ruichen Zhen, Rui Shao, Xiu Su, Zeke Xie, Hongxun Yao, and Shuo Yang. Learning to accelerate vision-language-action models through adaptive visual token caching, 2026. URL https://arxiv.org/abs/2602.00686.

Zhijie Wu, Kento Kawaharazuka, and Kei Okada. Neural introspection gating for adaptive kv-cache reuse in vision-language-action models, 2026. URL https://arxiv.org/abs/2608.10824.

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, 2025. URL https://arxiv.org/abs/2502.02175. arXiv:2502.02175.

Yuhao Xu, Yantai Yang, Zhenyang Fan, Yufan Liu, Yuming Li, Bing Li, and Zhipeng Zhang. Qvla: Not all channels are equal in vision-language-action model's quantization, 2026. URL https://arxiv.org/abs/2602.03782.

Jingxuan Zhang, Yunta Hsieh, Zhongwei Wan, Haokun Lin, Xin Wang, Ziqi Wang, Yingtie Lei, and Mi Zhang. Quantvla: Scale-calibrated post-training quantization for vision-language-action models, 2026. URL https://arxiv.org/abs/2602.20309.

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, 2025. URL https://arxiv.org/abs/2410.04417.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Ya-Qin Zhang, Jiangmiao Pang, Jingjing Liu, Tai Wang, and Xianyuan Zhan. X-vla: Soft-prompted transformer as scalable cross-embodiment vision-language-action model, 2025. URL https://arxiv.org/abs/2510.10274.

A LIMITATION AND DISCUSSION

We also acknowledge two limitations. First, LPA-CACHE targets the prefill-stage language-backbone, so its end-to-end benefit is inherently limited for models in which this stage accounts for only a small fraction of the total inference latency, such as X-VLA (Zheng et al., 2025). Second, the reuse-entry layers and task-relevance probe layers are currently selected empirically based on validation experiments. A more principled understanding of how these choices interact with cross-modal aggregation and task performance could further reduce manual tuning and broaden the applicability of the method.

B FORMULATING PREFILL-STAGE LANGUAGE-BACKBONE LATENCY

Consider a language backbone with L decoder layers and a full prefill sequence length N. Let $b_1 < \cdots < b_J$ denote the reuse entry layers, and let r_j denote the number of additional reused visual tokens at layer b_j. The active length at layer ℓ is

$$m_\ell = N - \sum_{j=1}^{J} r_j \, \mathbf{1}\{\ell \geq b_j\}, \qquad \sum_{j=1}^{J} r_j \leq |\mathcal{V}| - n. \tag{7}$$

For convenience, define the cumulative target active prefill lengths

$$M_0 = N, \qquad M_j = N - \sum_{k=1}^{j} r_k, \quad j = 1, \ldots, J. \tag{8}$$

The active prefill length is hence:

$$m_\ell = M_j \qquad \text{for } b_j \leq \ell < b_{j+1}. \tag{9}$$

That is, the first b_1 layers run at full length N, the next $b_2 - b_1$ layers run at length M_1, and so on.

Let $T_\ell(m; N)$ denote the latency of the ℓth decoder layer with m active query positions and a full-length KV sequence of size N, and given schedule $(\{b_j\}, \{M_j\})$, the language-backbone prefill latency is

$$\mathcal{L}_{\text{prefill}} = \sum_{\ell=0}^{L-1} T_\ell(m_\ell; N) = \sum_{j=0}^{J} \sum_{\ell=b_j}^{b_{j+1}-1} T_\ell(M_j; N). \tag{10}$$

Consider the representative latency profile $T(M_j; N)$ as introduced in equation 1, equation 10 simplifies to

$$\mathcal{L}_{\text{prefill}} \approx \sum_{j=0}^{J} (b_{j+1} - b_j)\, T(M_j; N). \tag{11}$$

Setting $J = 0$, or equivalently $M_j = N$ for all j, recovers the no-reuse baseline.

C INFERENCE PROCEDURE DEFINITION

The execution details of LPA-CACHE's inference procedure is as follows:

Algorithm 1 LPA-CACHE Inference Procedure

Input: Current observation x_t; language instruction l; protected-token count n.
Reuse schedule $\mathcal{R} = \{(b_j, M_j)\}_{j=1}^{J}$; probe-layer set $\mathcal{S}$.
Initialized previous-step state: e_{t-1}, A_{t-1}, $\{K_{t-1}^{(\ell)}, V_{t-1}^{(\ell)}\}$, and C_{t-1} (UniVLA only).
Let $M_0 = N$, with $N - M_J \le |\mathcal{V}| - n$, where $\mathcal{V}$ denotes visual positions.
Output: Robot action a_t; updated layerwise KV caches; e_t, A_t, and C_t (UniVLA only).

1: **Visual encoding.** Encode x_t to obtain visual embeddings e_t; retrieve cached e_{t-1} and construct the full input hidden-state sequence.
2: **Token protection.** Average previous-step task-relevance scores over probe layers $\mathcal{S}$; protect the top-n visual positions as $\mathcal{I}_{t-1}$.
$$\mathcal{I}_{t-1} = \mathrm{TopK}_{i \in \mathcal{V}}(\bar{a}_{t-1,i}, n).$$
3: **Change computation.** For each non-protected visual position $i \in \mathcal{V} \setminus \mathcal{I}_{t-1}$, compute the relative embedding change:
$$\delta_{t,i} = \frac{\|e_{t,i} - e_{t-1,i}\|_2}{\max(\|e_{t-1,i}\|_2, \varepsilon)}, \qquad \varepsilon > 0.$$
4: **Token ranking.** Sort non-protected visual positions in ascending order of $\delta_{t,i}$ to obtain the reuse-priority list π_t.
5: **Reuse assignment.** At each entry layer b_j, assign the next $r_j = M_{j-1} - M_j$ positions in π_t to reuse, so that the remaining active length is M_j.
6: **Selective-KV prefill.** For each decoder layer $\ell = 0, \ldots, L-1$:
– At a reuse-entry layer, remove its assigned visual positions from the active hidden-state stream; retain original sequence indices.
– Recompute and update KV entries only for active positions; retain cached entries elsewhere. Apply attention with m_ℓ queries and N keys/values under the model's attention mask.
– Keep protected tokens and all non-visual positions active. Record $A_t^{(\ell)}$ at probe layers; use FlashAttention-2 in non-probe layers.
7: **Final hidden states.** Apply final normalization to obtain compact hidden states $\widetilde{H}_t$ and retain their original-position mapping p_t.
8: **UniVLA adaptation.** Update the final-hidden-state cache at active positions; leave all other rows unchanged:
$$C_t[p_{t,k}, :] \leftarrow \widetilde{H}_t[k, :], \qquad k = 1, \ldots, m_{L-1}.$$
9: **Action generation.** Generate a_t using the model's original action interface. For UniVLA, provide $C_t[\nu, :]$ as the visual-feature readout in its original shape and order.
10: **Return.** Return a_t and save updated KV caches, e_t, A_t, and C_t (UniVLA only) for the next timestep.

D DETAILED CALVIN SUBTASKS SUCCESS RATES

Subtask	UniVLA	+ FlashAttn-2	+ VLA-CACHE	+ LPA-CACHE
rotate_blue_block_left	96.9%	96.9%	84.4%	87.1%
open_drawer	98.9%	99.4%	98.8%	97.7%
lift_pink_block_table	88.0%	89.5%	94.4%	93.5%
place_in_drawer	100.0%	98.8%	100.0%	98.8%
turn_on_led	96.9%	98.9%	100.0%	99.0%
lift_blue_block_table	95.4%	95.6%	96.4%	94.0%
move_slider_right	99.3%	100.0%	98.5%	100.0%
turn_off_lightbulb	92.8%	94.1%	85.9%	90.4%
push_red_block_left	95.7%	91.5%	89.4%	95.5%
push_pink_block_right	68.8%	59.4%	62.5%	67.9%
turn_off_led	100.0%	100.0%	100.0%	98.6%
move_slider_left	99.2%	97.6%	99.2%	99.2%
push_into_drawer	79.0%	74.1%	76.2%	76.3%
close_drawer	100.0%	100.0%	100.0%	100.0%
rotate_red_block_right	75.7%	76.3%	65.7%	76.3%
rotate_red_block_left	100.0%	100.0%	100.0%	88.5%
lift_blue_block_slider	83.9%	81.7%	80.0%	80.0%
push_blue_block_right	56.2%	54.5%	47.1%	48.5%
stack_block	61.4%	61.5%	53.6%	58.6%
unstack_block	96.3%	89.7%	91.7%	88.9%
push_blue_block_left	97.3%	94.3%	94.3%	94.3%
lift_red_block_slider	78.1%	86.4%	80.6%	85.5%
place_in_slider	91.2%	91.1%	91.1%	91.6%
turn_on_lightbulb	100.0%	100.0%	98.9%	98.8%
lift_red_block_table	93.3%	92.1%	88.5%	94.0%
rotate_blue_block_right	83.8%	86.1%	80.0%	81.1%
rotate_pink_block_left	82.6%	87.5%	91.7%	95.8%
lift_pink_block_slider	78.8%	95.4%	85.0%	77.4%
push_red_block_right	62.9%	67.6%	61.1%	52.8%
rotate_pink_block_right	81.8%	68.8%	87.9%	75.0%
push_pink_block_left	90.0%	87.2%	84.6%	97.5%
lift_pink_block_drawer	100.0%	77.8%	77.8%	100.0%
lift_red_block_drawer	100.0%	100.0%	100.0%	100.0%
lift_blue_block_drawer	100.0%	100.0%	100.0%	88.9%