VLA-ZO: FAST ZEROTH-ORDER ADAPTATION FOR VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Adapting vision-language-action (VLA) models to deployment-time distribution shifts is important for reliable robotic operation, but conventional first-order adaptation can exceed the memory budget of inference-oriented deployment platforms. Zeroth-order (ZO) optimization offers a forward-only alternative with inference-level memory, but accurate gradient estimation requires many perturbation queries, making naive ZO prohibitively slow for large VLA models. We present VLA-ZO, a framework for fast ZO adaptation that exploits the structure of VLA computation. By confining adaptation to the action side, VLA-ZO keeps the expensive vision-language prefix frozen and reuses its conditioning states across perturbation queries and optimizer steps, while schedule-aware prefetching hides state-transfer overhead. On LIBERO camera-viewpoint shifts, VLA-ZO reduces end-to-end adaptation time by 25.59× at $q = 16$ and 32.54× at $q = 64$ relative to baseline ZO, while improving average task success from 48.27% without adaptation to 58.17% and 63.58%, respectively. These results show that making ZO faster can make larger query budgets practical, providing a promising path toward resource-efficient VLA adaptation on deployment platforms.

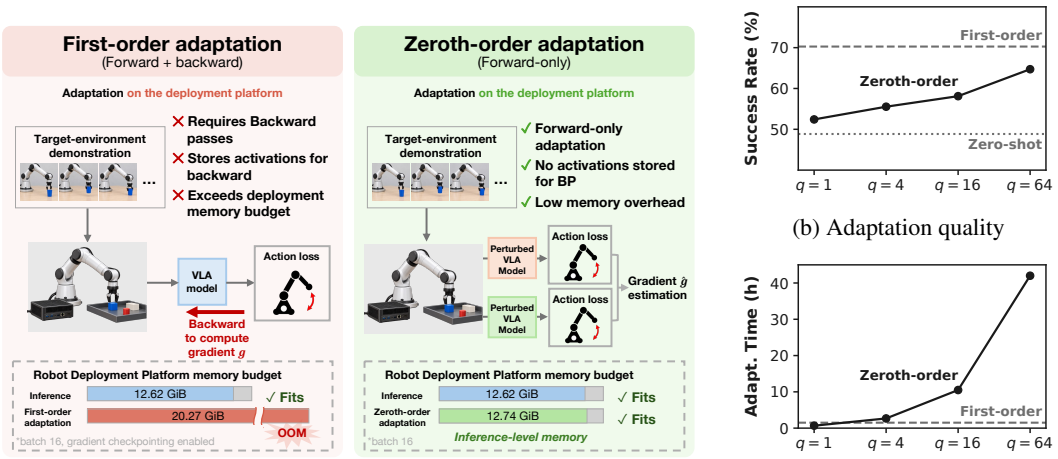
1 INTRODUCTION

Vision-Language-Action (VLA) models are emerging as a general interface for robotic control (Kim et al., 2024; Black et al., 2025b;a), but deploying a fixed policy is often insufficient in real-world environments. Changes in camera viewpoint, appearance, and operating conditions shift the observation distribution away from that seen during training, causing VLA policies to degrade sharply under such shifts. Specifically, the success rates of state-of-the-art VLAs collapse from 95% to below 30% under modest perturbations to camera pose and robot initial state (Fei et al., 2025). Recent work has explored one-shot adaptation (Li et al., 2026; Kang et al., 2026), in which a VLA policy is adapted using a single demonstration from a target environment. This setting enables data-efficient post-deployment adaptation to address these distribution shifts.

This adaptation, however, must run on the deployment platform itself, under a compute and memory budget provisioned for inference rather than training. Conventional first-order adaptation fits this setting poorly (Figure 1a, left): backpropagation requires retaining intermediate activations that can otherwise be discarded during inference, substantially increasing memory requirements (Cai et al., 2020; Yang et al., 2023). Even with gradient checkpointing enabled, this overhead grows with batch size: for $\pi_{0.5}$, first-order adaptation uses 20.27 GiB of peak memory at batch size 16 and 54.69 GiB at batch size 64, compared with the corresponding inference-level footprints of 12.74 GiB and 27.71 GiB for forward-only ZO. The relative gap therefore increases from $1.59\times$ to $1.97\times$. Moreover, gradient checkpointing trades memory for additional recomputation and does not remove the need for a backward-capable execution path. These requirements make first-order adaptation difficult to accommodate in inference-oriented deployment stacks as VLA models scale.

This motivates zeroth-order (ZO) optimization as a natural alternative (Figure 1a, right), as it enables adaptation using only forward evaluations without backpropagation. Rather than explicitly computing gradients through backpropagation, ZO methods estimate gradients by perturbing model parameters and comparing the resulting objective values using only forward evaluations (Duchi et al., 2015; Nesterov & Spokoiny, 2017; Malladi et al., 2023). By estimating parameter updates solely from forward evaluations, ZO avoids backpropagation and substantially reduces the mem-

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.



(a) Adaptation on the robot deployment platform

(b) Adaptation quality

(c) Adaptation time

Figure 1: **Zeroth-order adaptation enables on-robot adaptation within the inference memory budget.** Unlike first-order adaptation, it requires only forward evaluations and avoids storing activations (a). Increasing the query budget q improves adaptation quality toward first-order performance, at the cost of increased adaptation time (b,c).

ory requirements for adaptation, making it attractive for memory-constrained devices as well as deployment environments where backward execution is unavailable or undesirable. Despite these advantages, the potential of ZO optimization for adapting VLA policies remains largely unexplored.

In this work, we first show that ZO optimization can effectively adapt VLA policies under deployment-time distribution shifts, but at a substantial runtime cost from repeated forward evaluations (Figure 1). This trade-off is governed by the query budget q, the number of random perturbation directions evaluated per optimization step, each as a $\pm$ pair of forward passes. A larger q improves the gradient estimate, allowing adaptation quality to approach first-order performance while peak memory remains at the inference level. The same knob, however, also determines the computational cost: every additional direction requires two more forward passes through the full model, causing adaptation time to grow with q and quickly exceed that of first-order adaptation. Because modern VLAs contain expensive vision and language components, naively applying ZO can therefore require hours of adaptation. Thus, the main obstacle to practical ZO-based VLA adaptation is not whether it works, but whether it can run fast enough.

2 Related Work

VLA adaptation. Recent work has increasingly studied data-efficient adaptation of pretrained VLA policies from limited target-domain supervision (Li et al., 2025; Zhang et al., 2026; Xu et al., 2026; Nguyen et al., 2026). At the extreme of this low-data regime, one-shot methods adapt a VLA policy using only a single target-domain demonstration. FTM and FLA enable such adaptation to novel camera viewpoints (Li et al., 2026), while DART addresses visual and embodiment shifts and considers domain-level adaptation across tasks (Kang et al., 2026). Our focus is complementary: rather than developing a new adaptation method, we study how to reduce the computational cost of zeroth-order VLA adaptation under deployment-time compute and memory constraints.

Zeroth-order optimization. Zeroth-order (ZO) optimization estimates gradients from perturbed forward evaluations without backpropagation. MeZO (Malladi et al., 2023) enables LLM fine-tuning with approximately the memory footprint of inference and reports performance comparable to backpropagation on several language tasks. Subsequent benchmarking examines how task alignment and optimizer design affect ZO fine-tuning (Zhang et al., 2024), while SubZero explores low-rank perturbations in random subspaces (Yu et al., 2025). For execution efficiency, MobiZO parallelizes perturbation evaluations through Multi-Perturbed LoRA within existing inference engines (Gao et al., 2025). In general deep-network training, DeepZero combines sparse coordinate-wise gradient esti-

mation with reuse of features preceding the perturbed layer and forward parallelization (Chen et al., 2024).

3 ZEROTH-ORDER VLA ADAPTATION

3.1 PROBLEM SETUP AND ZO WORKLOAD

We study post-deployment adaptation of a pretrained VLA policy from a single target-domain demonstration, a common one-shot VLA adaptation setting used in recent studies (Li et al., 2026; Zhao et al., 2025; Kang et al., 2026). Following the domain-level setting of Kang et al. (2026), the demonstration is collected from one task in the target domain, and the adapted policy is evaluated on both the demonstrated task and unseen tasks in the same domain. For adaptation, we use one target-domain demonstration trajectory per scene, $\mathcal{D} = \{(o_i, \ell, a_i)\}_{i=1}^{N}$, where N is the number of observation–action pairs, o_i denotes the i-th observation (camera images and proprioceptive state), ℓ is the language instruction shared across the trajectory, and a_i is the corresponding action chunk.

To characterize the computation overhead during ZO adaptation, we separate the VLA policy into a *vision-language prefix* f_ϕ with parameters ϕ and an *action expert* g_ψ with parameters ψ. Let π_θ, with $\theta = (\phi, \psi)$, denote the VLA policy. For each sample $(o_i, \ell, a_i) \in \mathcal{D}$, the frozen prefix first forms a complete, sample-specific conditioning state h_i; the action expert then uses h_i to define the policy over action chunks:

$$\begin{aligned} h_i &= f_\phi(o_i, \ell) \qquad \text{(vision-language prefix)}, \\ \pi_\theta(a_i \mid o_i, \ell) &= g_\psi(a_i \mid h_i) \quad \text{(action expert or head)}. \end{aligned} \tag{1}$$

Here, h_i denotes the conditioning state passed to the action module, and g_ψ denotes the action expert that maps h_i to a chunk of low-level commands. In the $\pi_{0.5}$ architecture, h comprises the prefix's attention keys and values together with the mask marking its valid positions, whereas in OpenVLA-OFT, h_i is represented by the action-token hidden state. This separation is common to recent VLA designs, including OpenVLA-OFT (Kim et al., 2025), π_0 (Black et al., 2025b), $\pi_{0.5}$ (Black et al., 2025a), SmolVLA (Shukor et al., 2025), and the GR00T family (Bjorck et al., 2025). In all of them, the prefix accounts for the majority of parameters and of the floating-point operations of a single forward evaluation of the adaptation objective.

Let λ denote the trainable adaptation parameters, which induce adapted model weights $\theta(\lambda)$, with $\theta(0) = \theta$ recovering the base policy. For sample i, let $\ell_i(\lambda; \xi)$ denote the action loss of the adapted policy, where ξ collects any stochastic variables required by the action objective. For $\pi_{0.5}$, ξ consists of an action-space noise sample and a flow time. The adaptation objective is

$$\mathcal{L}(\lambda) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_\xi \left[\ell_i(\lambda; \xi) \right]. \tag{2}$$

Zeroth-order optimization replaces the gradient of Eq. (2) with a randomized finite-difference estimate (Duchi et al., 2015; Nesterov & Spokoiny, 2017). Let $\mathcal{L}_\mathcal{B}(\lambda; \xi)$ denote the same objective averaged over a minibatch $\mathcal{B}$ of B samples drawn from $\mathcal{D}$, and evaluated at a fixed draw $\xi = (\xi_i)_{i \in \mathcal{B}}$. With q perturbation directions $z_j \sim \mathcal{N}(0, I_{d_\lambda})$, where d_λ is the dimension of λ and I_{d_λ} is the corresponding identity matrix, perturbation scale ϵ, and a random draw ξ_j for the minibatch associated with each direction j,

$$\hat{\nabla}\mathcal{L}(\lambda) \;=\; \frac{1}{q} \sum_{j=1}^{q} \frac{\mathcal{L}_\mathcal{B}(\lambda + \epsilon z_j;\, \xi_j) - \mathcal{L}_\mathcal{B}(\lambda - \epsilon z_j;\, \xi_j)}{2\epsilon}\, z_j, \tag{3}$$

where the two evaluations of a $\pm$ pair share the draw ξ_j, and $\mathcal{B}$ is held fixed across all q directions of an update. An update therefore consumes $2q$ objective evaluations (equivalently, $2qB$ sample-level action evaluations) and never constructs a backward graph. Two properties of this workload determine what an implementation can reuse: *which* parameters adaptation has to place under the perturbation (Section 3.2), and *what* each of the $2qB$ evaluations is forced to recompute as a result (Section 3.3).

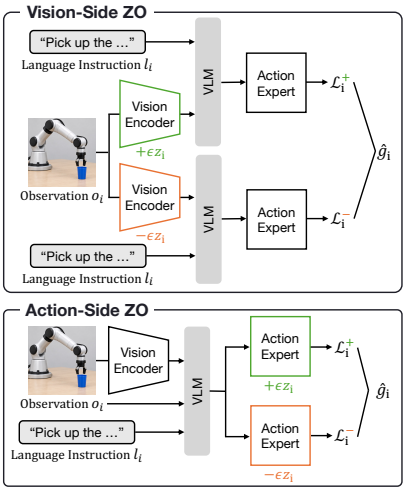
3.2 ACTION-SIDE ADAPTATION WITH A FROZEN PREFIX

Prior VLA adaptation methods often adapt vision-side parameters, including vision encoders, visual representations, and visual projection layers (Kim et al., 2024; Grover et al., 2026; Abouzeid et al., 2025; Li et al., 2026). Among these, Feature Linear Adaptation (FLA) (Li et al., 2026) adapts the vision encoder with low-rank updates while keeping the downstream language and action components frozen. We instead ask whether ZO adaptation can be confined to the action expert while keeping the expensive vision-language prefix frozen.

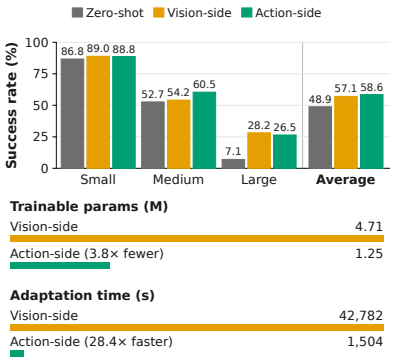
Figure 2a compares two LoRA placements for $\pi_{0.5}$. The *vision-side* placement targets the feed-forward projections of every block of the vision encoder inside f_ϕ, following the placement used by Li et al. (2026). The *action-side* placement targets the query and value projections of every self-attention block of the action expert g_ψ. At the same rank $r = 16$, the action-side adapter contains 1.25M trainable parameters, compared with 4.71M for the vision-side adapter. Both configurations keep the pretrained model weights frozen and optimize only their LoRA parameters with ZO using the objective in Eq. (2).

Under this matched setup, Figure 2b shows that action-side ZO increases average success across the three camera-viewpoint shifts from 48.9% to 58.6%, while matching vision-side ZO on average (57.1%). Given that it uses approximately 3.8× fewer trainable parameters, this result motivates confining adaptation to the action expert while keeping the vision-language prefix frozen.

This makes action-side adaptation a more efficient choice in our method, achieving comparable performance with substantially shorter adaptation time, in addition to requiring fewer trainable parameters. This efficiency stems from the prefix output being independent of the perturbed parameters. For a fixed demonstration sample, its conditioning state is therefore invariant across ZO probes and optimizer steps, exposing the redundancy analyzed next.



(a) ZO adaptation placement.



(b) Viewpoint-shift experiment results.

Figure 2: **Action-side ZO improves success with a frozen prefix.**

3.3 Repeated Prefix Computation

Thus, each h_i needs to be computed at most once and can be reused whenever the same sample is revisited, reducing the number of prefix executions from $2qBT$ to at most N. In contrast, the $2qBT$ action-expert evaluations remain necessary because the action-expert parameters change across ZO probes.

Under action-side adaptation, the prefix parameters ϕ remain fixed, while only the action-expert parameters depend on the adaptation parameters λ, i.e., $\theta(\lambda) = (\phi, \psi(\lambda))$. Consequently, for each demonstration sample i, the conditioning state $h_i = f_\phi(o_i, \ell)$ is independent of λ. A naive implementation recomputes the vision-language prefix for every one of these evaluations. However, the demonstration contains only N distinct observations $\{o_i\}_{i=1}^N$, and each observation has a fixed conditioning state h_i because ϕ is frozen. The required $2qBT$ action-expert evaluations cannot be removed, but the number of prefix executions can in principle be reduced from $2qBT$ to at most N if each distinct conditioning state is computed once, cached, and reused thereafter.

The remaining systems challenge is to store and supply these states efficiently to the action expert, which Section 4 addresses through within-step reuse, across-step reuse, and schedule-aware prefetch.

4 VLA-ZO: Reusing Frozen Conditioning Across ZO Probes

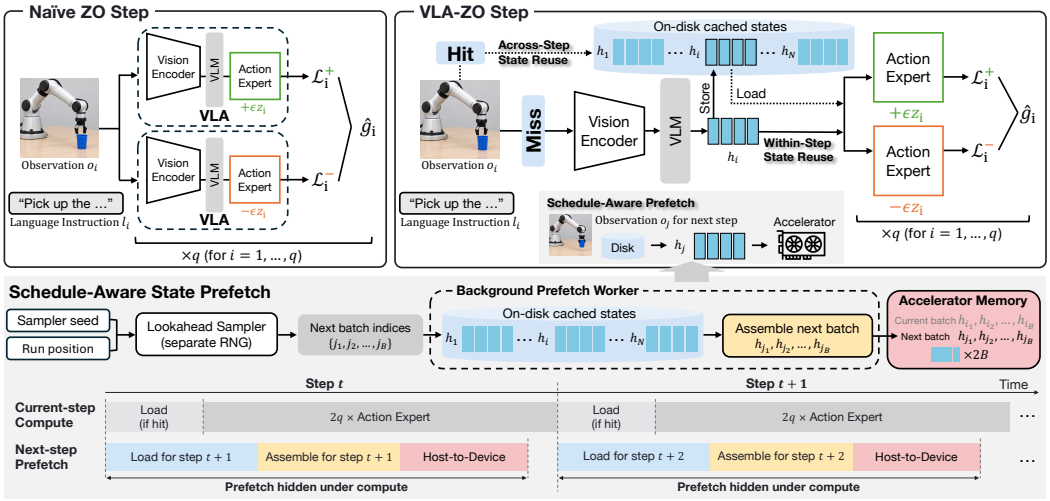


Figure 3: Overview of VLA-ZO.

VLA-ZO aims to accelerate ZO adaptation without changing its objective, perturbation directions, or trainable parameter space. The key observation is that once adaptation is confined to the action expert, the expensive vision-language prefix becomes independent of the ZO perturbation. In a naive ZO step, each of the $2q$ perturbed evaluations recomputes the same frozen-prefix state for a sample, resulting in $2qBT$ prefix executions over T updates despite only N distinct conditioning states. VLA-ZO addresses this computational bottleneck in ZO adaptation through three optimizations, illustrated in Figure 3. First, within an update step, it computes the prefix once per sample and reuses the resulting state across all $2q$ probes (Section 4.1). Second, the same prefix states are retained across update steps, so that each distinct observation requires only one prefix evaluation over the full adaptation run (Section 4.2). Finally, VLA-ZO overlaps transfer of the next cached batch with the current probe evaluations using schedule-aware prefetching (Section 4.3).

Cost model. Let c_{pre} and c_{suf} denote the per-sample costs of the frozen prefix and action suffix, respectively, and let c_{ld} denote the per-sample cost of loading a cached conditioning state. For comparison, we separate the recurring per-update cost from the one-time cost of materializing the N distinct conditioning states, which is Nc_{pre}. The resulting per-update costs are

$$C_{\mathrm{naive}} = B\big(2q\,c_{\mathrm{suf}} + 2q\,c_{\mathrm{pre}}\big), \tag{4a}$$

$$C_{\mathrm{within}} = B\big(2q\,c_{\mathrm{suf}} + c_{\mathrm{pre}}\big), \tag{4b}$$

$$C_{\mathrm{across}} = B\big(2q\,c_{\mathrm{suf}} + c_{\mathrm{ld}}\big), \tag{4c}$$

$$\text{and} \quad C_{\mathrm{prefetch}} = B\big(2q\,c_{\mathrm{suf}} + \max(0,\ c_{\mathrm{ld}} - 2q\,c_{\mathrm{suf}})\big). \tag{4d}$$

Here, C_{naive} denotes the per-update cost of naive execution, while C_{within}, C_{across}, and C_{prefetch} denote the costs after progressively adding within-step reuse, across-step reuse, and schedule-aware prefetch, respectively. The common term $2qBc_{\mathrm{suf}}$ is the action-expert work required by the ZO estimator in Eq. (3). The three mechanisms below progressively reduce the remaining prefix-related overhead: first by eliminating repeated prefix execution, then by reusing cached prefix states across update steps, and finally by hiding cache-transfer latency behind action-expert computation.

4.1 Within-Step State Reuse

Within a ZO update, all $2q$ probes for a given sample share the same input (o_i, ℓ). Because adaptation is confined to the action expert, perturbing the adaptation parameters changes only the action expert, while the frozen prefix produces the same conditioning state h_i for every probe. VLA-ZO computes

$h_i = f_\phi(o_i, \ell)$ once per sample and reuses it across all $2q$ perturbed action-expert evaluations. For a minibatch of size B, this reduces the number of prefix evaluations per update from $2qB$ to B, while leaving the required $2qB$ action-expert evaluations unchanged. Accordingly, the per-update cost decreases from Eq. (4a) to Eq. (4b).

The mathematical equivalence of reuse and recomputation rests on three conditions, all of which are met by our cached action-side implementations. First, all perturbed parameters must lie after the reuse boundary: we adapt the action expert in $\pi_{0.5}$ and the action head in OpenVLA-OFT. Second, h_i must contain the complete information passed from the prefix to the action expert: $\pi_{0.5}$ caches the prefix attention state and padding mask, while OpenVLA-OFT caches the action-token hidden features consumed by its action head. Third, the frozen prefix must be deterministic for a fixed (o_i, ℓ) under the execution configuration used for caching. Thus, under exact arithmetic, reuse leaves the objective and finite-difference estimator unchanged.

Effect of the query budget. Without reuse, each additional perturbation direction adds two full-policy evaluations per sample, with cost $2(c_{\text{pre}} + c_{\text{suf}})$. With within-step reuse, it adds only two action-expert evaluations, with cost $2c_{\text{suf}}$. Relative to naive execution, the per-update speedup of within-step reuse is

$$\frac{C_{\text{naive}}}{C_{\text{within}}} = \frac{2q(c_{\text{pre}} + c_{\text{suf}})}{c_{\text{pre}} + 2qc_{\text{suf}}}. \tag{5}$$

As q increases, the prefix cost is amortized over more probes and the speedup approaches $(c_{\text{pre}} + c_{\text{suf}})/c_{\text{suf}}$, the cost ratio between a full forward evaluation and an action-expert-only evaluation. This is useful in the large-q regime where ZO obtains a better gradient estimate but naive execution becomes increasingly expensive.

4.2 Across-Step State Reuse

Within-step reuse still recomputes the prefix whenever a sample appears in a new optimizer step. In one-shot adaptation, however, every minibatch is drawn from the same demonstration of N observations, and the prefix remains frozen for the entire run. Therefore, once h_i has been computed for a sample, the same state can be reused every time that sample is revisited. VLA-ZO stores each conditioning state after its first computation and looks it up on subsequent visits. The cache is populated lazily, so a miss runs the prefix once and stores the result, while a hit skips prefix execution entirely. Over T updates, within-step reuse performs BT prefix evaluations, whereas ideal across-step reuse requires at most one prefix evaluation for each of the N demonstration observations. All adaptation times reported in Section 5 include the cost of populating the cache on first visits.

Once all N demonstration observations have been encountered, the cumulative execution costs over T updates are

$$\begin{aligned} C_{\text{within}}(T) &= BT\left(c_{\text{pre}} + 2qc_{\text{suf}}\right), \\ C_{\text{across}}(T) &= Nc_{\text{pre}} + (BT - N)c_{\text{ld}} + 2qBTc_{\text{suf}}. \end{aligned} \tag{6}$$

Effect of repeated visits. Across-step reuse replaces repeated prefix executions with cached-state loads. Relative to within-step reuse, the cumulative cost reduction over T update steps is

$$C_{\text{within}}(T) - C_{\text{across}}(T) = (BT - N)\left(c_{\text{pre}} - c_{\text{ld}}\right). \tag{7}$$

Thus, the benefit grows with the number of repeated sample visits, $BT - N$, while the one-time cost of materializing the N distinct states is amortized over more updates. As discussed in Section 3.1, the vision-language prefix accounts for the majority of the forward-pass FLOPs in most VLA architectures. Consequently, loading a cached conditioning state is substantially cheaper than recomputing the prefix, with $c_{\text{ld}} \ll c_{\text{pre}}$.

4.3 Schedule-Aware State Prefetch

Across-step reuse replaces repeated prefix execution with cached-state loading, but keeping all N conditioning states resident on the accelerator would make device memory grow with the demonstration size. VLA-ZO therefore stores the persistent cache on disk and stages only the states required by each minibatch. This bounds accelerator-side cache storage, but places disk loading, batch assembly, and host-to-device transfer on the critical path if performed synchronously. VLA-ZO hides

this latency by overlapping preparation of the next cached batch with the $2q$ probe evaluations of the current update, as illustrated in Figure 3.

Schedule-aware lookahead. Let $\mathcal{B}_t$ denote the minibatch used at update t and let κ_t denote the corresponding sampler position. For a fixed sampler seed s, the minibatch schedule is determined by the sampling rule, $\mathcal{B}_t = \mathcal{S}(s, \kappa_t)$. Thus, while update t is running, VLA-ZO can reconstruct $\mathcal{B}_{t+1}$ from (s, κ_{t+1}) using a separate local generator. This lookahead follows the same permutation and batch-formation rules as the live sampler, but does not advance its state or consume randomness used for the perturbations or objective.

For cached samples in $\mathcal{B}_{t+1}$, a background worker reads the conditioning states, assembles the next batch, and transfers it to the accelerator while the current batch evaluates its $2q$ action-expert probes. The worker never invokes the policy or materializes cache misses; an incompletely cached future batch remains on the ordinary demand path when that update begins. We prefetch only one update ahead, so device staging contains at most the current and next batches of conditioning states. Implementation details are given in Appendix B.

Effect of latency hiding. For a fully cached minibatch, loading its conditioning states costs Bc_{ld}, while evaluating the $2q$ action-expert probes costs $2qBc_{\mathrm{suf}}$. Without prefetching, these costs are serialized, yielding $C_{\mathrm{across}} = B(c_{\mathrm{ld}} + 2qc_{\mathrm{suf}})$. With one-update lookahead, loading for the next update overlaps with probe computation for the current update. After pipeline warm-up, the recurring per-update cost is therefore determined by the slower of the two:

$$\begin{aligned} C_{\mathrm{prefetch}} &= \max(Bc_{\mathrm{ld}},\, 2qBc_{\mathrm{suf}}) \\ &= B\left(2qc_{\mathrm{suf}} + \max(0, c_{\mathrm{ld}} - 2qc_{\mathrm{suf}})\right), \end{aligned} \tag{8}$$

recovering Eq. (4d). Thus, the loading latency that remains exposed on the critical path is $B\max(0, c_{\mathrm{ld}} - 2qc_{\mathrm{suf}})$. Loading is fully hidden whenever $c_{\mathrm{ld}} \le 2q\, c_{\mathrm{suf}}$, or, equivalently, when $q \ge \lceil c_{\mathrm{ld}}/(2c_{\mathrm{suf}}) \rceil$. Thus, increasing the query budget also lengthens the suffix-computation window available to hide cache I/O. Once the condition above holds, the recurring update cost reduces to $2qBc_{\mathrm{suf}}$, and the speedup relative to naive execution reaches $(c_{\mathrm{pre}} + c_{\mathrm{suf}})/c_{\mathrm{suf}}$.

5 Experiments

5.1 Setup

Benchmark and Evaluation We evaluate $\pi_{0.5}$ (Black et al., 2025a) and OpenVLA-OFT (Kim et al., 2025) on all 40 tasks in the LIBERO Spatial, Object, Goal, and LIBERO-10 suites (Liu et al., 2023). Following DART's scene-wise one-shot protocol (Kang et al., 2026), adaptation uses five expert demonstrations, one per scene, rendered under the target visual condition. We test three camera shifts (`Small`, `Medium`, and `Large`) and compound shifts combining `Medium` with image blur or altered lighting. Each condition is evaluated over 40 initial states per task (1,600 episodes), using indices 10–49 and including tasks without adaptation demonstrations. Initial-state indices 0–9 are excluded because they were used for learning-rate selection.

Baselines and Ablations We compare VLA-ZO with the unadapted policy, FLA (Li et al., 2026), and DART (Kang et al., 2026). Our FLA and DART baselines, denoted by †, use zeroth-order gradient estimates computed using only forward passes and update the parameters with SGD. At each query budget, the execution ablation fixes the adaptation settings and progressively enables within-step reuse, across-step reuse, and schedule-aware prefetch. Appendix A provides the data protocol, perturbations, model-specific rollout settings, hyperparameters, and timing procedure.

5.2 Camera-Viewpoint Adaptation

Table 1 evaluates two aspects of VLA-ZO: whether adapting the action expert improves robustness to a changed viewpoint, and how much time is saved by executing a fixed ZO workload with progressively more reuse. The zero-shot, FLA, and DART rows provide adaptation-quality reference points; the four action-side rows at each q provide a cumulative ablation of the execution mechanisms.

Table 1: **Camera-viewpoint adaptation of $\pi_{0.5}$.** At fixed q, successive action-side rows add execution optimizations under identical adaptation settings. Times include cold-cache population; speedups are relative to baseline ZO at the same q. † denotes our ZO-SGD baseline implementations.

		Novel viewpoint (success rate, %)				Adaptation		
Method	q	Small	Medium	Large	Average	Params (M)	Time (s)	Speedup
Zero-shot	–	86.19	51.44	7.19	48.27	–	–	–
FLA† (Li et al., 2026)	16	88.56	53.31	27.81	56.56	4.71	42,782	–
DART† (Kang et al., 2026)	16	89.50	58.88	14.25	54.21	3,353.43	43,794	–
Baseline ZO	16	86.44	60.69	25.75	57.63	1.25	38,487	1.00×
+ Within-step prefix reuse	16	88.00	60.31	25.00	57.77	1.25	2,474	15.56×
+ Across-step prefix reuse	16	88.06	60.88	25.56	58.17	1.25	2,264	17.00×
+ Schedule-aware prefetch (VLA-ZO)	16	88.50	59.81	26.19	58.17	1.25	1,504	25.59×
Baseline ZO	64	87.31	67.00	37.00	63.77	1.25	153,220	1.00×
+ Within-step prefix reuse	64	87.56	66.38	36.63	63.52	1.25	5,734	26.72×
+ Across-step prefix reuse	64	87.19	65.75	35.56	62.83	1.25	5,465	28.04×
+ Schedule-aware prefetch (VLA-ZO)	64	87.56	66.38	36.81	63.58	1.25	4,709	32.54×

Table 2: **Compound visual shifts with $\pi_{0.5}$, by LIBERO suite.** Avg. is the mean across LIBERO suites. † denotes our ZO-SGD baseline implementations.

		View+Noise					View+Light					Adaptation	
Method	q	Spa.	Obj.	Goal	10	**Avg.**	Spa.	Obj.	Goal	10	**Avg.**	Params (M)	Time (s)
Zero-shot	–	50.5	84.0	42.0	20.3	49.2	39.8	86.8	53.8	15.8	49.0	–	–
FLA† (Li et al., 2026)	16	61.8	70.8	47.0	29.0	53.0	43.5	73.5	57.5	35.3	52.4	4.71	43,081
DART† (Kang et al., 2026)	16	57.5	90.8	45.5	27.8	55.4	52.3	92.3	50.8	31.8	56.8	3,353.43	43,965
VLA-ZO	16	46.0	86.8	43.5	28.0	51.1	60.0	90.3	56.3	27.0	58.4	1.25	1,500
VLA-ZO	64	49.0	91.5	49.3	33.3	55.8	68.5	89.0	59.0	26.8	60.8	1.25	4,726

Adaptation quality and query budget. The action-side adapter also improves average success with fewer trainable parameters than either ZO baseline. At $q = 16$, VLA-ZO reaches 58.17% average success, compared with 48.27% for the unadapted policy. At $q = 64$, its average rises to 63.58%. The larger budget helps most under the `Medium` and `Large` shifts, where success increases from 59.81% to 66.38% and from 26.19% to 36.81%, respectively. The `Small` condition already has high zero-shot success and does not improve consistently with q. Thus, additional perturbation directions are most useful in the harder viewpoint conditions in this experiment.

Execution time and the contribution of reuse. At $q = 16$, the full system reduces adaptation time from 38,487 to 1,504 seconds ($25.59\times$). At $q = 64$, it reduces time from 153,220 to 4,709 seconds ($32.54\times$). Within-step reuse accounts for the largest reduction, bringing the two workloads to 2,474 and 5,734 seconds. Across-step reuse then lowers these times to 2,264 and 5,465 seconds, and prefetching supplies the final reduction. This ordering is consistent with the cost model in Eq. (4): first remove prefix execution from individual probes, then reuse conditioning across updates, and finally overlap its loading with action-expert computation.

Quality across execution variants. At fixed q, reuse changes when conditioning is computed and transferred, while preserving the optimization problem. The observed average-success ranges are 57.63–58.17% at $q = 16$ and 62.83–63.77% at $q = 64$. These variations should not be interpreted as quality improvements or degradations from caching. Split prefix/suffix execution can change floating-point kernels, and small loss differences can accumulate through finite differences and successive updates (Section B.4). The table supports similar observed adaptation quality.

Table 3: **Camera-viewpoint adaptation of OpenVLA-OFT.** Success rates (%) are reported by viewpoint severity and averaged across severities. Action-side adaptation targets the separate L1 regression head. The reported VLA-ZO timing includes cold feature-cache population. † denotes our ZO-SGD baseline implementations.

		Novel viewpoint (success rate, %)				Adaptation	
Method	q	`Small`	`Medium`	`Large`	Average	Params (M)	Time (s)
Zero-shot	–	80.19	50.88	17.38	49.48	–	–
FLA† (Li et al., 2026)	4	81.06	52.63	19.31	51.00	4.71	18,283
DART† (Kang et al., 2026)	4	81.13	53.63	20.50	51.75	7,709.17	28,482
VLA-ZO	4	79.06	57.06	21.88	52.67	0.79	109
VLA-ZO	16	79.63	55.69	21.25	52.19	0.79	170

Table 4: **Compound visual shifts with OpenVLA-OFT, by LIBERO suite.** Avg. is the mean across LIBERO suites. † denotes our ZO-SGD baseline implementations.

		`View+Noise`					`View+Light`					Adaptation	
Method	q	Spa.	Obj.	Goal	10	**Avg.**	Spa.	Obj.	Goal	10	**Avg.**	Params (M)	Time (s)
Zero-shot	–	17.0	41.0	17.3	20.3	23.9	26.3	86.8	26.0	24.5	40.9	–	–
FLA† (Li et al., 2026)	4	21.8	47.3	14.0	21.8	26.2	26.5	87.3	25.8	28.3	41.9	4.71	18,283
DART† (Kang et al., 2026)	4	29.8	43.3	18.0	24.8	28.9	30.8	86.0	24.8	26.8	42.1	7,709.17	28,265
VLA-ZO	16	29.3	45.8	17.5	24.3	29.2	31.5	88.0	28.8	26.8	43.8	0.79	170
VLA-ZO	64	31.8	65.5	13.3	23.5	33.5	34.8	89.3	32.8	28.3	46.3	0.79	403

5.3 Compound Visual Shifts

Tables 2 examines whether the benefits extend beyond camera geometry. Each compound condition retains the `Medium` viewpoint and adds blur or altered lighting. Under `View+Noise`, VLA-ZO reaches 51.1% average success at $q = 16$ and 55.8% at $q = 64$, compared with 49.2% without adaptation. Under `View+Light`, the corresponding averages are 58.4% and 60.8%, compared with 49.0% without adaptation. The gains are larger for the lighting condition. At $q = 64$, VLA-ZO is close to the DART baseline on `View+Noise` and exceeds it on `View+Light`.

5.4 Generalization to OpenVLA-OFT

OpenVLA-OFT places a separate L1 action-regression head after the frozen vision-language model. It evaluates reuse at a different architectural boundary from the flow-matching action expert of $\pi_{0.5}$.

Table 3 shows that action-head adaptation at $q = 4$ raises average viewpoint-shift success from 49.48% to 52.67%. The improvement is concentrated in the `Medium` and `Large` conditions, while `Small` changes little. Table 4 further shows that the benefit extends to compound visual shifts. Under `View+Noise`, average success increases from 23.9% without adaptation to 29.2% at $q = 16$ and 33.5% at $q = 64$. Under `View+Light`, it increases from 40.9% to 43.8% and 46.3%, respectively. Together with the viewpoint results, these results show that the action-head adaptation strategy transfers to OpenVLA-OFT.

6 Conclusion

We presented VLA-ZO, a fast forward-only framework for one-shot VLA adaptation that eliminates redundant computation across zeroth-order queries. More broadly, our results highlight a unique opportunity in ZO adaptation: speed can translate directly into adaptation quality, because faster execution enables more query directions and thus better gradient estimates within a fixed time budget. This suggests fast ZO as a promising path toward practical VLA adaptation under resource-constrained deployment platforms.

AI USE STATEMENT

In this work, we used generative AI tools to assist with drafting portions of the manuscript, improving the clarity and readability of the writing, and searching for and identifying potentially relevant literature. All AI-assisted text was reviewed and revised by the authors. References and related work suggested through AI-assisted retrieval were independently verified by the authors against the original sources before inclusion in the paper. The authors take responsibility for the final content of this work, including all text, claims, citations, and other artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We describe the zeroth-order adaptation formulation in Section 3.1, the VLA-ZO design and execution mechanisms in Section 4, and the evaluation setup in Section 5. Additional details on the adaptation data protocol, visual shifts, model and rollout settings, hyperparameters, timing methodology, and the implementation of state reuse and prefetching are provided in Appendices A and B. Code and experimental artifacts will be released upon acceptance.

REFERENCES

Ali Abouzeid, Malak Mansour, Qinbo Sun, Zezhou Sun, and Dezhen Song. Geoaware-vla: Implicit geometry aware vision-language-action model. *arXiv preprint arXiv:2509.14117*, 2025.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, brian ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*, 2025a. URL https://openreview.net/forum?id=vlhoswksBO.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert 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, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025b. doi: 10.15607/RSS.2025.XXI.010.

Han Cai, Chuang Gan, Ligeng Zhu Massachusetts Institute of Technology, and Song Han Massachusetts Institute of Technology. Tinytl: reduce memory, not parameters for efficient on-device learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Aochuan Chen, Yimeng Zhang, Jinghan Jia, James Diffenderfer, Konstantinos Parasyris, Jiancheng Liu, Yihua Zhang, Zheng Zhang, Bhavya Kailkhura, and Sijia Liu. Deepzero: Scaling up zeroth-order optimization for deep model training. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=qBWhjsNPEY.

John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806, 2015.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. Libero-plus: In-depth robustness analysis of vision-language-action models, 2025. URL https://arxiv.org/abs/2510.13626.

Lei Gao, Amir Ziashahabi, Yue Niu, Salman Avestimehr, and Murali Annavaram. MobiZO: Enabling efficient LLM fine-tuning at the edge via inference engines. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20206–20223, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1022. URL https://aclanthology.org/2025.emnlp-main.1022/.

Shresth Grover, Akshay Gopalkrishnan, Bo Ai, Henrik I. Christensen, Hao Su, and Xuanlin Li. Enhancing generalization in vision language action models by preserving pretrained representations. In *International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, Sep 2026. IEEE/RSJ.

Taewook Kang, Taeheon Kim, Donghyun Shin, and Jonghyun Choi. Domain arithmetic: One-shot vla adaptation under environmental shifts. In *Computer Vision – ECCV 2026*, pp. 423–441. Springer Nature Switzerland, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R 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. In *8th Annual Conference on Robot Learning*, 2024. URL https://openreview.net/forum?id=ZMnD6QZAE6.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025. doi: 10.15607/RSS.2025.XXI.017.

Puhao Li, Yingying Wu, Ziheng Xi, Wanlin Li, Yuzhe Huang, Zhiyuan Zhang, Yinghan Chen, Jianan Wang, Song-Chun Zhu, Tengyu Liu, and Siyuan Huang. ControlVLA: Few-shot object-centric adaptation for pre-trained vision-language-action models. In *9th Annual Conference on Robot Learning*, 2025. URL https://openreview.net/forum?id=kXhOmN3x18.

Weiqi Li, Quande Zhang, Ruifeng Zhai, Liang Lin, and Guangrun Wang. Vla models are more generalizable than you think: Revisiting physical and spatial modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 35025–35035, June 2026.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, qiang liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=xzEtNSuDJk.

Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. Fine-tuning language models with just forward passes. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Vota6rFhBQ.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. PiSSA: Principal singular values and singular vectors adaptation of large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=6ZBHIEtdP4.

Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

Minh Duc Nguyen, Nghiem Tuong Diep, Nguyen Gia Binh, Trong-Bao Ho, Doanh Le Thien, Quang Tan Nguyen, Thien-Loc Ha, Tran Van Nhiem, Bao Thach, Tran Xuan Nhat, Tuan Anh Tran, Artur Habuda, Philip Lund Møller, Tran Nguyen Le, Daniel Sonntag, Mathias Niepert, Khoa D Doan, Vu N. Duong, Hung Ngo, Minh Nhat VU, Duy Minh Ho Nguyen, An Thai Le, and Vien Anh Ngo. FOCA: Future-oriented conditioning for data-efficient vision-language-action adaptation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=0cnDTttyMR.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

Changhua Xu, En Yu, Junyu Xuan, and Jie Lu. Vgas: Value-guided action-chunk selection for few-shot vision-language-action adaptation. In Diego Calvanese (ed.), *Proceedings of the Thirty-Fifth International Joint Conference on Artificial Intelligence, IJCAI-26*, pp. 5110–5118. International Joint Conferences on Artificial Intelligence Organization, 8 2026. doi: 10.24963/ijcai.2026/569. URL https://doi.org/10.24963/ijcai.2026/569. Main Track.

Yuedong Yang, Guihong Li, and Radu Marculescu. Efficient on-device training via gradient filtering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3811–3820, June 2023.

Ziming Yu, Pan Zhou, Sike Wang, Jia Li, Mi Tian, and Hua Huang. Zeroth-order fine-tuning of llms in random subspaces. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4475–4485, October 2025.

Yang Zhang, Chenwei Wang, ouyang lu, Yuan Zhao, Yunfei Ge, Zhenglong Sun, Xiu Li, Chi Zhang, Chenjia Bai, and Xuelong Li. Align-then-steer: Adapting the vision-language action models through unified latent guidance. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=T3i7Ifeatk.

Yihua Zhang, Pingzhi Li, Junyuan Hong, Jiaxiang Li, Yimeng Zhang, Wenqing Zheng, Pin-Yu Chen, Jason D. Lee, Wotao Yin, Mingyi Hong, Zhangyang Wang, Sijia Liu, and Tianlong Chen. Revisiting zeroth-order optimization for memory-efficient LLM fine-tuning: A benchmark. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=THPjMr2r0S.

Ruihan Zhao, Tyler Ingebrand, Sandeep Chinchali, and Ufuk Topcu. Mos-vla: A vision-language-action model with one-shot skill adaptation. *arXiv preprint arXiv:2510.16617*, 2025.

A Experimental Protocol and Hyperparameters

A.1 DART Adaptation Data Protocol

Scene-wise supervision and transfer across tasks. We adopt the task and demonstration selection protocol in Appendix C.2 and Table 13 of DART (Kang et al., 2026). The four evaluation suites partition 40 tasks, whereas the five scenes define the one-shot supervision budget. Table 5 lists the first task combination, used in the experiments here. For condition-specific adaptation, its five trajectories are pooled into one dataset. Evaluation includes all 40 tasks, so the policy must transfer the adaptation to tasks outside this set of five demonstration tasks. The demonstrations retain their task instructions and actions across camera views.

Table 5: Adaptation task combination 1 from DART's Table 13. Lengths count retained observation-action pairs after no-op removal.

Scene	Demonstrated task	Length
Living room	Put both the alphabet soup and the tomato sauce in the basket.	336
Kitchen	Turn on the stove and put the moka pot on it.	272
Study	Pick up the book and place it in the back compartment of the caddy.	234
Floor	Pick up the alphabet soup and place it in the basket.	145
Tabletop	Pick up the black bowl between the plate and the ramekin and place it on the plate.	98

DART defines three task combinations. The retained dataset reconstructs all three, containing 1,085, 1,044, and 1,025 frames per camera view, respectively. We use combination 1 and seed 0.

Selecting the first demonstration. The source is the filtered `openvla/modified_libero_rlds` dataset, which DART uses for its demonstration selection. The first demonstration is the lowest original per-task demonstration ID still present in this filtered release, rather than the first serialized TFRecord encountered. This distinction matters because shard order is shuffled and filtering removes some original demonstrations. The implementation identifies each eligible episode by matching its float32 action sequence to the original LIBERO demonstration after applying the same no-op removal rule. It then selects the lowest eligible ID. For example, the living-room trajectory in combination 1 is `demo_1`, because `demo_0` is absent from the filtered source. The selected actions for combination 1 match the released `SNUMPR/dart-libero-viewpoints` data; the public release itself contains only this combination.

A.2 Visual Perturbations

Table 6 specifies the external-camera translations. The simulator also changes camera orientation to look toward the scene origin, so the rotation depends on the original scene-specific camera pose. The wrist camera is not moved by this viewpoint transformation.

The condition named `Noise` applies OpenCV Gaussian blur with a 27×27 kernel and automatically inferred standard deviation (`sigmaX=0`) to the external image. The implementation does not blur the wrist image. `Light` loads the modified LIBERO scene assets with stronger blue illumination and changed overhead-light positions; scene lighting affects both camera views. Compound conditions apply these changes together with the `Medium` camera pose. Matching perturbed adaptation bundles preserve the demonstration actions, task selection, and scene protocol. Action-side and FLA adapters are trained separately for each condition. DART uses a matching target stage and the canonical source stage. Thus, these experiments measure adaptation with target-condition demonstrations rather than merely testing a clean-view adapter on an additional corruption.

A.3 Models, Action Windows, and Evaluation

Table 7 summarizes the rollout settings. The replanning interval is the number of actions executed from each predicted chunk before querying the policy with a new observation. Maximum control horizons exclude the initial settling steps and are held fixed within each model's comparisons.

Table 6: Camera translations relative to the original external camera, following the pinned LIBERO-view implementation of DART's viewpoint shifts.

Severity	Δx (m)	Δy (m)	Δz (m)
`Small`	0.0	0.3	-0.1
`Medium`	-0.2	0.7	-0.2
`Large`	-1.2	1.0	-0.2

Table 7: Model-specific LIBERO rollout settings. The relation to released model settings and evaluation defaults is specified below.

Setting	$\pi_{0.5}$	OpenVLA-OFT
Predicted action chunk length	50	8
Actions executed before replanning	10	8
Maximum control steps: Spatial	220	220
Maximum control steps: Object	280	280
Maximum control steps: Goal	300	300
Maximum control steps: LIBERO-10	520	520
Initial settling steps	10	10
Episodes used for LR selection	10 (0–9)	10 (0–9)
Evaluation episodes per task	40 (10–49)	40 (10–49)
Rendered camera resolution	256×256	256×256
Policy image resolution	224×224	224×224

$\pi_{0.5}$ model. The checkpoint is `lerobot/pi05_libero_finetuned_v044`. We retain the released checkpoint's 50-action chunks, ten flow-integration steps at inference, and pretrained observation/action preprocessing and normalization. Images are rendered at 256×256, matching the checkpoint's declared camera inputs, and resized internally to 224×224. The ten-action replanning interval follows LeRobot's documented LIBERO evaluation command; this overrides the checkpoint's stored `n_action_steps=50`. The per-suite rollout limits, benchmark initial-state selection, and ten settling steps match LeRobot v0.6.1's LIBERO defaults. The adaptation objective is the native flow-matching loss. Action noise and flow time are resampled between perturbation directions and replayed within each positive/negative pair. All directions in an update use the same effective minibatch.

OpenVLA-OFT model. The checkpoint is `moojink/openvla-7b-oft-finetuned-libero-spatial-object-goal-10`. We follow the official LIBERO evaluation defaults for the continuous L1-regression head, two camera inputs with proprioception, eight-action replanning, per-suite rollout limits, ten settling steps, and 50 episodes per task. We also retain its 256×256 rendering and official image preprocessing to 224×224 with center cropping. The eight-action training chunks and checkpoint-provided, suite-specific `BOUNDS_Q99` normalization follow the released LIBERO model configuration. This action-head loss does not sample a flow time or action noise.

Evaluation and aggregation. For both models, we use 50 benchmark initial states per task, reserving indices 0–9 for learning-rate selection and evaluating the final models on indices 10–49. This yields 40 evaluation episodes per task, 400 per suite, and 1,600 episodes in total for each condition. For $\pi_{0.5}$, this increases the ten episodes per task in LeRobot's documented evaluation protocol to 50; for OpenVLA-OFT, it matches the official default. We set the experiment seed to 0 for both models. Success rates weight tasks equally; suite and condition averages use the groups specified in each result table. Training and policy serving run separately from the LIBERO simulator, with observations and action chunks exchanged between the two processes. No simulator rollout is needed for a ZO training loss evaluation.

Table 8: Selected $\pi_{0.5}$ adaptation configurations. † denotes our ZO-SGD baseline implementations. DART's two stages each start from the same pretrained weights. Execution ablations reuse the action-side settings at the corresponding q.

Method	Trainable placement	q	Learning rate	Updates
Action-side ZO	Action-expert query/value LoRA	16	3×10^{-2}	1,000
Action-side ZO	Action-expert query/value LoRA	64	10^{-1}	1,000
FLA†	Vision encoder FFN LoRA	16	10^{-3}	1,000
DART†	Full model, source and target	16	10^{-3}	$500 + 500$

Table 9: Selected OpenVLA-OFT adaptation configurations. Action-side LoRA targets the three eligible linears of the L1 action head, and FLA targets the SigLIP FFNs of the fused vision backbone. DART trains all VLA, action-head, and proprio-projector parameters; its two stages each start from the same pretrained weights. Compound-shift runs reuse these settings with matching perturbed demonstrations.

Method	Trainable placement	q	Learning rate	Updates
Action-side ZO	Action-head LoRA	4	10^{-2}	1,000
FLA†	Vision encoder FFN LoRA	4	3×10^{-3}	1,000
DART†	Full model, source and target	4	3×10^{-5}	$500 + 500$

A.4 Hyperparameters and Learning-Rate Sweeps

Adaptation methods and execution ablations. We compare the unadapted policy with action-side ZO adaptation and ZO-SGD implementations of FLA (Li et al., 2026) and DART (Kang et al., 2026). The daggered baselines replace the original first-order optimization with two-sided MeZO estimates followed by SGD. FLA adapts vision-encoder feed-forward projections; DART adapts source- and target-domain full-model weights separately and combines their updates using domain arithmetic. Both change the prefix and therefore cannot use our frozen-prefix cache. At fixed q, the execution ablation holds the adaptation data, initialization, learning rate, parameter placement, and update count fixed, progressively adding within-step prefix reuse, persistent reuse across steps, and schedule-aware prefetch to uncached action-side ZO.

Fixed optimization settings. Tables 8 and 9 summarize the $\pi_{0.5}$ and OpenVLA-OFT settings. All listed stages use task combination 1, seed 0, batch and microbatch size 64, Gaussian two-sided perturbations, and $\epsilon = 10^{-3}$. The ZO update averages q directional estimates and applies SGD without momentum, weight decay, or gradient clipping. LoRA uses rank and scaling 16, exact-SVD PiSSA initialization (Meng et al., 2024), and trains both factors.

Learning rate sweep. Every OpenVLA-OFT sweep, including the FLA-style and DART baselines, uses the half-decade learning-rate grid $\{10^{-5}, 3\times10^{-5}, 10^{-4}, 3\times10^{-4}, 10^{-3}, 3\times10^{-3}, 10^{-2}, 3\times10^{-2}, 10^{-1}\}$. Each completed candidate receives a pilot evaluation on the first ten initial states of every task, or 400 episodes per view.

A.5 Timing and Software

The recorded profiling environment uses NVIDIA RTX PRO 6000 Blackwell Server Edition GPUs with 96 GB memory, with one GPU visible to each training process.

DART uses two 500-update training windows and a separately timed CPU merge. The merge timer covers checkpoint validation and loading, domain arithmetic, output serialization, and hashing. Full adaptation accounts for source training, target training, and this merge; when a source checkpoint is reused across target conditions, target-only adaptation excludes the shared source stage.

B Implementation of Frozen-State Reuse and Prefetching

This appendix specifies the cache boundary, minibatch schedule, and transfer protocol behind Section 4.3.

B.1 Cache Contents

The reuse boundary of $\pi_{0.5}$. For $\pi_{0.5}$, a cached conditioning state contains the attention keys and values from every frozen vision–language layer, together with the prefix padding mask. For a single observation, each layer's key and value tensors have shape $[1, H_{\mathrm{KV}}, L, d_h]$, and the Boolean mask has shape $[1, L]$. Here, L is the padded prefix length, H_{KV} the number of key/value heads, and d_h the head dimension. This state contains the information required by the action expert from the frozen prefix. Action targets, sampled action noise, flow time, and trainable action projections remain outside the cache.

The reuse boundary of OpenVLA-OFT. For OpenVLA-OFT, a cached conditioning state contains the final-layer hidden states of the frozen language model at the action-token positions. For a single observation, this tensor has shape $[1, KD_a, d]$, and the L1 head reshapes it to $[1, K, D_a d]$ before its first projection. Here, $K = 8$ is the action-chunk length, $D_a = 7$ the action dimension, and $d = 4096$ the language-model width. Both camera images, the proprioceptive state, and the prompt are processed before this boundary by the frozen vision backbone, projectors, and language model, and the action head reads no other input; the state therefore contains all information the head requires from the frozen VLA.

Cold and mixed minibatches. Cache construction is lazy and takes place in the training thread. If every sample in a microbatch is cached, its states are loaded in sampler order and prefix computation is skipped. If any record is absent, the implementation executes the frozen prefix at the original full microbatch shape. It writes only new records and checks that previously stored rows match the recomputed values exactly. Keeping the original shape avoids changing matrix-multiplication and attention kernels by compacting only the missing samples. Consequently, a mixed microbatch can recompute cached observations during cache population: there is at most one stored record per observation and namespace, but there can be more than one physical prefix evaluation of that observation. The Nc_{pre} population term in Section 4.2 describes ideal per-observation materialization; this conservative fallback can add cold-cache computation. Missing files fall back to demand-time computation, whereas corrupt records, incompatible layouts, changed inputs, or detected numerical mismatches raise an error.

B.2 Reconstructing the Next Minibatch

Let b be the microbatch size and a the number of accumulated microbatches, so that the effective batch size is $B = ab$. For update t, the training thread first obtains all a microbatches and their cached or freshly computed prefix states. It then schedules the a microbatches of update $t+1$ before evaluating the current update's $2q$ probes. The queue is bounded to one future optimizer update. Only fully cached future microbatches are eligible; incomplete ones remain on the ordinary demand path. A future that is not ready when requested delays its own batch and never changes the sample order. The implementation requires standard ordered batching and the configured full microbatch shape. The consumed cursor, optimizer state, and training random-number state are checkpointed; speculative futures are rebuilt after resume, and persistent files can either be reused or repopulated lazily.

B.3 Host Staging and CUDA Streams

The cache persists on disk, while a single background worker reads records and assembles a complete microbatch in reusable pinned host buffers. On a dedicated CUDA stream, the worker allocates device tensors with matching shapes, dtypes, and strides, enqueues nonblocking host-to-device copies, and records a completion event. Before consuming a staged batch, the compute stream waits on that event and records its use of the device tensors through `record_stream`. This establishes the copy-to-compute dependency without synchronizing the entire device. The host buffers remain

alive and unchanged until their copy event completes; only then can they return to the staging pool. The worker performs file reads, tensor assembly, and transfers only. All model calls, cache-miss computation, and objective randomness remain in the training thread.

B.4 What Reuse Costs and What It Preserves

What reuse costs. The price of all three mechanisms is memory, and it is bounded by the demonstration rather than by the workload. The accelerator staging budget covers two batches of conditioning states: the B the current update uses, and the B the prefetcher is staging for the next one. Their payload is therefore at most $2BS$, where S is the size of one conditioning state h_i and depends on how the interface between the frozen prefix and the action expert is represented in each architecture. This excludes the action expert's working tensors and cloned attention cache (Appendix B). For $\pi_{0.5}$, h_i consists of the prefix attention key–value states from 18 VLM layers and occupies approximately 17.02 MiB per sample in BF16. In contrast, OpenVLA-OFT caches a 56×4096 action-token hidden state, requiring only 0.4375 MiB per sample in BF16. On the host, the store grows to N records and no further, regardless of q or T.

What reuse preserves. VLA-ZO changes only the execution of ZO adaptation: it does not alter the objective, perturbation directions, trainable parameters, or update rule. For $\pi_{0.5}$, the cached conditioning state is computed from the frozen vision–language prefix and fixed inputs, so reusing it is neutral to the finite-difference estimator in exact arithmetic. Action noise and flow time remain sampled for each direction and are replayed within its $\pm$ pair. Bitwise-identical training trajectories are not implied: the cached path separates prefix prefill from action denoising and uses a different GPU execution path, while deterministic CUDA algorithms are not enforced. Small numerical differences in the two losses can be amplified over successive ZO updates. Empirically, our cache-mode ablation in Table 1 finds success-rate differences comparable to those between repeated executions of an identical configuration.