MAKING COMPUTATION COUNT: SPECULATIVE STATE RECYCLING FOR FLOW-BASED VLAS

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative inference accelerates flow-based vision-language-action models (Flow-VLAs) by bypassing vision-language model (VLM) prefill and full flow integration on speculative rounds. However, lightweight drafting and approximate verification can compromise action quality, while additional computation to improve reliability can erode the resulting speedup. Moreover, rejection can erase these savings by triggering full-path inference after the draft and verification costs have already been incurred. These limitations point to a broader inefficiency: speculative states are often discarded once they are no longer directly usable, even though they may still contain useful predictive information. We propose **Speculative State Recycling (SSR)**, which improves this quality–efficiency trade-off by recycling previously computed states across changes in conditioning. Before refresh, retained deep VLM features serve as semantic memory for a shallow predictor, augmenting the current observation with prior high-level representations. The same shallow computation also produces a draft-independent Context Risk estimate of how much refreshing the conditioning would change the full model's actions, guiding execution and refresh. When refresh is needed, saved proposals initialize short fresh-conditioned Action Expert integration, reducing regeneration from noise. On LIBERO, SSR with Triton nearly matches the success rate of full-generation $\pi_{0.5}$ with compiled inference while achieving a $1.89\times$ speedup per executed action and 74.1% fewer full AE calls per executed action. On LIBERO-Plus, SSR improves mean success by 5.77 points over FLASH at only 0.15 ms/action additional cost. Our project website is available at `https://ssracc.github.io/speculative-state-recycling/`.

1 INTRODUCTION

Flow-based vision-language-action models (Flow-VLAs) generate continuous action chunks from noise by integrating a velocity field conditioned on images, language, and robot state (Black et al., 2025b;a). Their high inference cost limits high-frequency real-time control, motivating approaches that reduce computation or overlap inference with action execution (Yang et al., 2025; Black et al., 2025c). In large language models (LLMs), speculative inference accelerates generation through lightweight drafting and parallel target-model verification, with suitable acceptance rules preserving the target distribution (Leviathan et al., 2023; Chen et al., 2023). This token-wise paradigm has also been extended to autoregressive VLAs, where actions are generated sequentially as discrete tokens (Kim et al., 2025; Wang et al., 2025a). Flow-VLAs differ fundamentally: they generate an action chunk jointly through sequential integration along flow time rather than along action-token positions, so token-prefix verification does not transfer directly.

FLASH (Niu et al., 2026) addresses this obstacle by reconstructing Action Expert (AE) endpoints in parallel: agreement between the reconstructed endpoints and the draft determines an executable prefix without complete flow integration. To avoid fresh VLM prefill on speculative rounds, FLASH reuses stale vision-language model (VLM) key–value (KV) conditioning to approximate verification under the current observation. To limit the error introduced by stale KV, FLASH periodically refreshes the VLM cache and invokes full inference when triggered by verification or phase rules; otherwise, it avoids both fresh VLM prefill and full sequential AE integration (Niu et al., 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

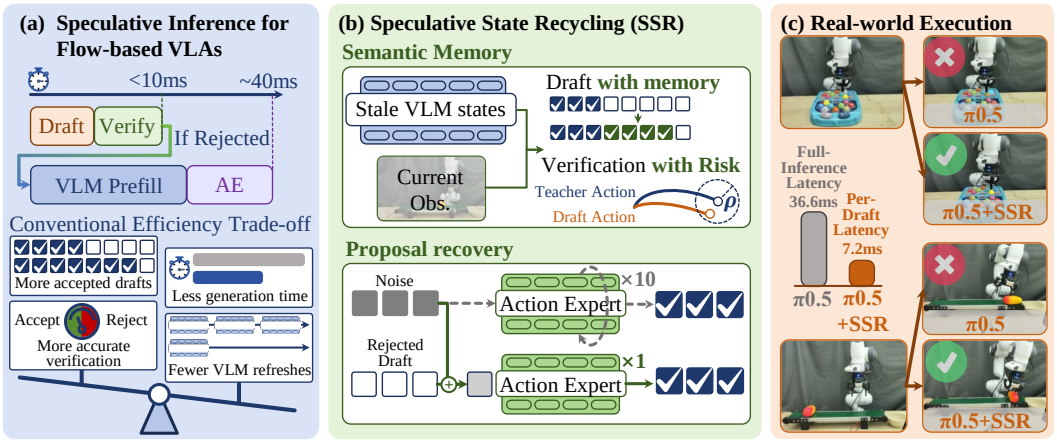


Figure 1: **Speculative State Recycling (SSR).** (a) Efficient speculation balances executable action yield against complete inference cost. (b) SSR reuses VLM features for drafting and Context Risk prediction, and saved proposals for recovery after refresh. (c) Real-robot examples with full-generation and SSR draft-path latencies; execution cycles are schematic.

However, reducing computation per speculative round does not necessarily reduce inference cost per executed action. A short accepted prefix spreads that computation over only a few executed actions, paralleling the dependence of LLM speculative speedup on acceptance and draft cost (Leviathan et al., 2023; Li et al., 2024). Improving drafting and verification can increase executable yield or reliability, but larger drafters raise proposal cost, while verification under fresh VLM conditioning adds prefill cost. Moreover, rejection can erase these savings: full-path fallback incurs VLM prefill and complete AE generation after draft and verification costs have already been paid (Niu et al., 2026). We therefore view Flow-VLA speculative efficiency as a trade-off between executable action yield and complete inference cost.

A key source of this inefficiency is that speculative states may still contain useful information after they are no longer directly usable. Prior work exploits target-model features for lightweight drafting in LLMs (Li et al., 2024) and reuses temporally redundant visual-token computations in VLAs (Xu et al., 2025). When combined with the latest observation, retained VLM features can better align lightweight predictions with the current fresh policy (Figure 4a) and support context-dependent refresh instead of relying only on a fixed schedule. Rejected proposals can likewise retain action structure useful for generation after conditioning is refreshed. These observations motivate **stateful computation recycling**: retaining useful information from previous computation and reusing it across changes in conditioning.

We introduce **Speculative State Recycling (SSR)** to apply this principle throughout speculative inference for Flow-VLAs (Figure 1). Before refresh, retained deep VLM features serve as **semantic memory** for a shared shallow predictor that supports both action drafting and **Context Risk** estimation. Together with action verification, Context Risk determines the executable prefix and whether cached conditioning should be refreshed. When refresh is required, saved proposals initialize short fresh-conditioned AE integration rather than restarting from noise, followed by displacement-bounded execution with full generation as fallback. Overall, SSR increases executable action yield relative to complete inference cost while preserving task quality.

In summary, our contributions are as follows: **(1) Output–cost analysis.** We formalize Flow-VLA speculative efficiency through executable action yield and complete inference cost, and identify discarded intermediate states as a source of wasted speculative computation. **(2) Speculative State Recycling.** We introduce SSR, which recycles retained semantic states for lightweight prediction and context-dependent refresh, and rejected proposals for short fresh-conditioned generation. **(3) Simulation and real-robot evaluation.** SSR with Triton nearly matches full-generation $\pi_{0.5}$ with compiled inference on LIBERO at a $1.89\times$ per-action speedup, improves LIBERO-Plus success over FLASH by 5.77 points at only 0.15 ms/action additional cost, and improves efficiency and task performance on real-robot tasks.

2 Related Work

2.1 Efficient VLA Inference

Recent work accelerates VLA inference by reducing model computation through quantization, pruning, lightweight architectures, and dynamic computation (Zhang et al., 2026; Yang et al., 2025; Wang et al., 2026; Wen et al., 2025; Li et al., 2026a). For diffusion- and flow-based policies, iterative action generation can also be shortened through distillation or few-step objectives (Wang et al., 2025b; Frans et al., 2025; Geng et al., 2025; Chen et al., 2026b). Other approaches improve real-time control by overlapping inference with action execution (Black et al., 2025c;d; Tang et al., 2026; Li et al., 2026b), or by optimizing caching, kernels, and system execution (Xu et al., 2025; Wei et al., 2026; Ma et al., 2025; Jiang et al., 2026). SSR instead operates within speculative inference, where acceleration requires not only reducing computation but also determining whether the cheaper speculative path remains sufficiently aligned with the target policy.

2.2 Speculative Inference

Speculative decoding accelerates autoregressive language models through lightweight drafting and parallel target-model verification (Leviathan et al., 2023; Chen et al., 2023). Later methods improve drafting through self-speculation, multi-head or feature-based prediction, and more parallel proposal generation (Cai et al., 2024; Zhang et al., 2024; Li et al., 2024; 2025; Gloeckle et al., 2024; Chen et al., 2026a; Cheng et al., 2026). Speculative inference has also been explored for diffusion models (De Bortoli et al., 2025; Hu et al., 2025).

For autoregressive VLAs, token-wise speculation remains applicable and has been adapted through relaxed or kinematics-aware verification (Wang et al., 2025a; Zheng et al., 2026a;b). Flow-VLAs instead generate continuous action chunks through flow-time integration. FLASH (Niu et al., 2026) introduces parallel Action Expert verification for this setting, while recent work also combines speculation with sparsity or hardware co-design (Liu et al., 2026; Qi et al., 2026). SSR builds on Flow-VLA speculative inference but focuses on recycling information that remains useful after cached context or rejected proposals are no longer directly usable.

3 Speculative Inference and Execution Efficiency

We first summarize FLASH's speculative inference path (Niu et al., 2026) and formulate efficiency in terms of executable action yield and complete inference cost. At round t, a Flow-VLA produces an action chunk $A_t = (a_{t,1}, \ldots, a_{t,H})$ from observation o_t, instruction ℓ, and robot state s_t. Full inference requires VLM processing followed by sequential AE integration over flow time (Lipman et al., 2023; Black et al., 2025b). FLASH instead verifies a lightweight proposal $\hat{A}_t$ using AE conditioning K_τ cached at the latest VLM refresh $\tau \le t$. With Gaussian noise ϵ_v, it reconstructs proposal endpoints in parallel at selected flow times $\sigma \in \Sigma_v$:

$$X^v_{t,\sigma} = \sigma \hat{A}_t + (1-\sigma)\epsilon_v, \qquad \widetilde{A}^v_{t,\sigma} = X^v_{t,\sigma} + (1-\sigma)v(X^v_{t,\sigma}, \sigma \mid K_\tau, s_t). \tag{1}$$

An action passes if all reconstructed endpoints remain within pose-distance threshold τ_A of the proposal, and $L^A_t \in \{0, \ldots, H\}$ denotes the longest consecutive passing prefix. This path avoids fresh VLM prefill and full AE integration, but agreement under cached conditioning need not imply agreement with the current fresh-conditioned policy. Per-round savings do not directly translate into low inference cost per executed action. A short accepted prefix amortizes speculative computation over few actions, while rejection adds fallback computation to work already spent. We therefore group prediction and any refresh or fallback before action output into one logical round. Let N_D, N_F denote executed actions from draft and full-generation rounds, and T_D, T_F their accumulated complete inference times. For draft-producing rounds $t \in \mathcal{S}$, let α_t be the executed fraction of the proposed chunk:

$$\mathcal{E}_{\mathrm{FLASH}} = \frac{N_D + N_F}{T_D + T_F} = \frac{H \sum_{t\in\mathcal{S}} \alpha_t + N_F}{T_D + T_F}. \tag{2}$$

Its inverse is the mean inference time per executed action $\bar{T}_{\mathrm{act}}$, evaluated together with task quality. To expose the main cost tradeoff, let n_D, n_F count draft and full-generation rounds and $n_{\mathrm{rej}} \le n_F$

count full-generation rounds following rejection. Using representative stage costs,

$$T_D \approx n_D(c_{\text{img}} + c_{\text{draft}} + c_{\text{verify}}),$$
$$T_F \approx n_F(c_{\text{img}} + c_{\text{prefill}} + c_{\text{AE}}) + n_{\text{rej}}(c_{\text{draft}} + c_{\text{verify}}). \tag{3}$$

Thus, improving speculative reliability can add computation, while a rejected round incurs speculative overhead before full generation. This motivates improving the utility of computation already produced during speculation, rather than relying on additional computation to improve reliability. We therefore evaluate subsequent designs by task quality and inference cost per executed action.

4 METHOD

Building on the output–cost analysis in Section 3, SSR recycles retained VLM features for lightweight prediction and context assessment, and rejected proposals for generation after refresh (Figure 2). We describe semantic memory (Section 4.1), Context Risk for execution and refresh (Section 4.2), and proposal recovery after refresh (Section 4.3).

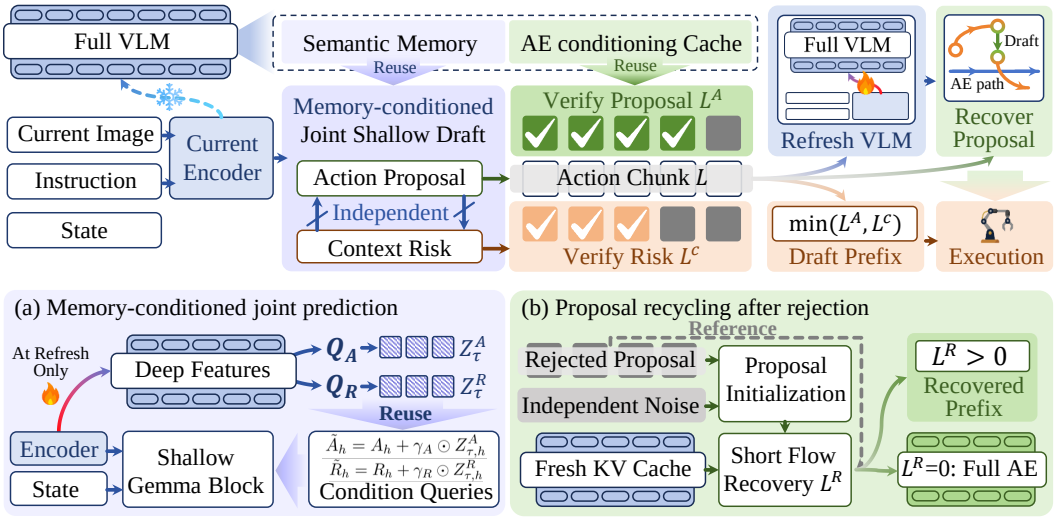


Figure 2: **Speculative State Recycling inference framework.** Retained VLM features support memory-conditioned joint prediction of action proposals and Context Risk, which together with Action verification determines the executable prefix and refresh decision. After refresh, rejected proposals initialize short fresh-conditioned recovery, with full generation as fallback.

4.1 SEMANTIC MEMORY FOR LIGHTWEIGHT PREDICTION

A shallow drafter must recover task-relevant information with far less computation than the full VLM. Since deep VLM features can encode shared task-relevant semantics across visual and linguistic inputs, we retain them as semantic memory for lightweight prediction. At each full refresh, SSR retains the deep hidden sequence $H_\tau \in \mathbb{R}^{N_\tau \times d}$ produced alongside the AE conditioning K_τ. The drafter still receives the current visual-language prefix $C_t = E_{\text{light}}(o_t, \ell)$ and current state embedding S_t. For each action position, a learned decoder query is first conditioned on the retained features and then attends to the current inputs in a single shallow Gemma (Gemma Team, 2024) block G. The shallow drafting backbone follows FLASH (Niu et al., 2026). Concretely, a query former Q_A uses learned memory-reader queries $U^A \in \mathbb{R}^{H \times d}$ to extract one memory vector per action position:

$$Z_\tau^A = Q_A(U^A, H_\tau), \qquad \tilde{A}_h = A_h + \gamma_A \odot Z^A_{\tau,h}. \tag{4}$$

Here $A_h \in \mathbb{R}^d$ is the learned decoder query, $\gamma_A \in \mathbb{R}^d$ is a learned channel-wise scale, and $\odot$ denotes elementwise multiplication. Because the memory residual is applied before attention, retained semantics can influence how each query attends to the current inputs. The resulting Action hidden

state $Y^A_{t,h}$ produces the proposal $\hat{A}_{t,h} = W_A Y^A_{t,h}$. Q_A runs only at refresh, and subsequent speculative rounds reuse Z^A_τ with updated current inputs. Thus, the drafter benefits from deep VLM features without repeating deep VLM computation or memory extraction. During training, it learns to reproduce fresh full-policy actions $A^\star_t$ using the Action loss $\mathcal{L}_A$ defined in Appendix A.3.

4.2 Context Risk for Execution and Refresh

Retained features can improve current drafting, but Action verification is still performed under cached AE conditioning K_τ. Even when a proposal passes, refreshing the conditioning may change the full model's actions. Context Risk predicts this change from current inputs and retained state, allowing SSR to decide how long to execute and when to refresh VLM conditions. We construct the training target from two full AE evaluations that share the same current robot state s_t and flow noise ϵ_t, differing only in conditioning:

$$A^{\text{stale}}_t = \mathcal{A}(\epsilon_t, s_t, K_\tau), \qquad A^{\text{fresh}}_t = \mathcal{A}(\epsilon_t, s_t, K_t), \tag{5}$$

where $\mathcal{A}$ denotes complete integration of the frozen AE. For each action position, the target $D_{t,h} \in \mathbb{R}^3$ measures position, rotation, and gripper discrepancies:

$$D_{t,h} = \left(\left\| A^{\text{stale}}_{t,h,\text{pos}} - A^{\text{fresh}}_{t,h,\text{pos}} \right\|_2, \left\| A^{\text{stale}}_{t,h,\text{rot}} - A^{\text{fresh}}_{t,h,\text{rot}} \right\|_2, \left| A^{\text{stale}}_{t,h,\text{grip}} - A^{\text{fresh}}_{t,h,\text{grip}} \right| \right). \tag{6}$$

Because only the conditioning changes, this target isolates context discrepancy from drafting error.

Context-dependent execution and refresh. At inference, the Risk branch predicts $\hat{D}_{t,h} \in \mathbb{R}^3$ for each action position. We normalize the three discrepancy channels by fixed validation-derived scales $\eta_c > 0$ and compare them under a tolerance λ. Since a later discrepancy need not invalidate earlier actions, SSR retains the longest prefix within this tolerance. For $\mathcal{C} = \{\text{pos}, \text{rot}, \text{grip}\}$,

$$\begin{aligned} r_{t,h} &= \max_{c \in \mathcal{C}} \frac{\hat{D}_{t,h,c}}{\eta_c}, \qquad L^C_t = \max\{l \in \{0, \dots, H\} : r_{t,h} \le \lambda \text{ for all } h \le l\}, \\ L^{\text{exec}}_t &= \min(L^A_t, L^C_t). \end{aligned} \tag{7}$$

L^A_t measures proposal consistency under cached conditioning, whereas L^C_t limits execution according to the predicted effect of context refresh. When $L^C_t = 0$, SSR refreshes the conditioning; otherwise, it caps the executable prefix without requiring immediate refresh. The threshold λ controls the tolerance for reusing cached context. With execution cap m_t, the draft fraction in Eq. (2) is $\alpha_t = \min(L^A_t, L^C_t, m_t)/H$, linking refresh tolerance directly to the task-quality–efficiency tradeoff.

Independent prediction with shared computation. To keep context assessment inexpensive, Action and Risk share the same shallow block G while using separate memory-conditioned queries and output heads. The Risk queries are constructed from H_τ analogously to the Action queries in Eq. (4), and their memory projections are likewise computed only at refresh and reused on subsequent speculative rounds. A single shallow forward then produces both predictions:

$$\begin{aligned} Y_t &= G([C_t \mid S_t \mid \tilde{A}_{1:H} \mid \tilde{R}_{1:H}]; \mathcal{M}), \\ \hat{A}_{t,h} &= W_A Y^A_{t,h}, \qquad \hat{D}_{t,h} = W_R Y^R_{t,h} \in \mathbb{R}^3. \end{aligned} \tag{8}$$

The attention mask $\mathcal{M}$ prevents the Action and Risk queries from reading each other while allowing both to share the current inputs. The Risk branch is trained with a Huber loss (Huber, 1964) on $D_{t,h}$. This objective updates only Risk-specific parameters; the shared block and Action branch are trained by the Action objective. Technical details are given in Appendix A.

4.3 Proposal Recycling after Rejection

When no executable prefix remains, full fallback discards the computation already spent on the proposal. Yet a rejected proposal may still retain action structure useful under fresh conditioning. SSR therefore saves eligible zero-prefix proposals, refreshes the conditioning, and uses them to initialize short AE integration. Recovered actions are executed only when they remain sufficiently close to the saved proposal; otherwise SSR falls back to full generation. Figure 3 illustrates the

Action- and Context-rejection routes and their shared refresh-and-recovery mechanism. Let $\hat{A}_{\text{src}}$ be the saved proposal and (K_f, s_f) the conditioning and state from the fresh observation. We mix the proposal with independent Gaussian noise ϵ_r at flow time $\sigma_0 \in (0, 1)$ and integrate the fresh-conditioned velocity field to the endpoint using $k \geq 1$ AE evaluations:

$$X_0 = \sigma_0 \hat{A}_{\text{src}} + (1 - \sigma_0)\epsilon_r, \qquad \Delta\sigma = \frac{1 - \sigma_0}{k}, \qquad \sigma_j = \sigma_0 + j\Delta\sigma,$$
$$X_{j+1} = X_j + \Delta\sigma\, v(X_j, \sigma_j \mid K_f, s_f), \quad j = 0, \ldots, k-1, \qquad A^{\text{raw}} = X_k. \tag{9}$$

Here σ_0 controls how strongly the initialization preserves the proposal, while k controls the number of fresh-conditioned AE evaluations. Because recovery may change different action positions by different amounts, SSR measures displacement from the saved proposal per position and retains the longest prefix within a bound. With serving-coordinate pose distance d_{act} and threshold $\rho > 0$,

$$d_h^R = d_{\text{act}}(A_h^{\text{raw}}, \hat{A}_{\text{src},h}), \qquad L^R = \max\{l \in \{0, \ldots, H\} : d_h^R \leq \rho \text{ for all } h \leq l\}. \tag{10}$$

A nonempty prefix is executed up to the execution cap. If $L^R = 0$ or recovery fails numerically, SSR runs full AE generation using the same fresh conditioning K_f, without another VLM prefill. Because recovery already uses fresh conditioning, the previous context prefix L_t^C no longer limits execution.

The displacement check requires no additional AE evaluation or Action verification. It bounds deviation from the saved proposal rather than certifying task correctness. Each refresh also updates the semantic memories used by subsequent speculative rounds; detailed cache and request handling are given in Appendix A.

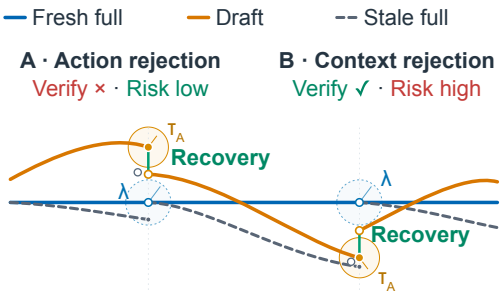


Figure 3: **Joint Action–Risk decisions and recovery.** Action rejection follows failed proposal verification (A), while context rejection follows high Context Risk despite Action consistency (B).

We evaluate recovery using the same complete-cost metric as Section 3, counting all speculative, recovery, and fallback computation in the logical round that ultimately produces the executed actions.

5 Experiments

5.1 Setup

We implement SSR on $\pi_{0.5}$ (Black et al., 2025a) and evaluate it in LIBERO simulation (Liu et al., 2023) and real-world tasks.

Simulation. We use the Spatial, Object, Goal, and Long (LIBERO-10) suites, each with 10 tasks and 50 trials per task using seed 7. A separate Action–Risk head is trained for each suite, with checkpoints selected by validation execution-prefix action error.

Inference. Full generation uses 10 AE steps, and each inference round executes at most 10 actions. Unless otherwise specified, SSR uses one recovery step ($k = 1$), a recovery bound $\rho = 0.15$, and an Action verification threshold of 0.15. We retain FLASH's gripper guard (Niu et al., 2026) and the post-verification truncation used in our baseline implementation. All experiments use RTX 4090 GPUs; simulation inference times and speedups are estimated from separately measured mean path times and recorded path usage. Additional details are given in Appendices B.4, B.5, and B.2.

5.2 LIBERO Evaluation

We compare SSR with full generation and FLASH (Niu et al., 2026). We use reported $\pi_{0.5}$ reproduction results as the full-generation reference. We report FLASH and SSR closed-loop results with compiled inference, and additionally report SSR with Triton. FLASH uses periodic refresh every

two speculative rounds (PF2). Our primary metrics are success rate (SR) and mean model inference time per executed action; VLM prefill and full AE generation (AE10) usage are reported to characterize inference paths.

Table 1: Success and inference cost on LIBERO. Time and usage are four-suite means. Calls are per 100 executed actions.

Methods	Success rate (%) ↑					ms/act.	Calls/100 act.		Full (%)	
	Goal	Object	Spatial	Long	Mean	↓	Prefill	AE10	Req.	Act.
Torch-$\pi_{0.5}$	**98.0**	98.2	**98.8**	**92.4**	**96.9**	4.39	10.00	10.00	100.0	100.0
FLASH-$\pi_{0.5}$	97.2	**99.6**	98.0	88.6	95.85	3.52	5.62	5.62	41.4	56.2
SSR-$\pi_{0.5}$	97.2	99.4	98.6	91.4	96.65	3.05	**4.44**	**2.51**	18.4	**24.8**
SSR+Triton-$\pi_{0.5}$	97.4	98.2	97.0	91.2	95.95	**2.32**	4.56	2.59	**18.4**	25.4

Standard LIBERO. On standard LIBERO, compiled SSR reaches 96.65% mean SR, close to the 96.9% full-generation reference and above FLASH at 95.85%, while reducing mean inference time from 3.52 to 3.05 ms/action relative to FLASH. With Triton, SSR reaches 95.95% SR at 2.32 ms/action. SSR also substantially reduces reliance on full generation, with only 24.8% of executed actions produced by AE10. Per-suite cost and usage are reported in Appendix B.3.

Table 2: Success and inference cost on LIBERO-Plus. Time and usage are four-suite means; metric definitions follow Table 1.

Methods	Success rate (%) ↑					ms/act.	Calls/100 act. ↓		Full act. (%)
	Goal	Object	Spatial	Long	Mean	↓	Prefill	AE10	↓
FLASH-$\pi_{0.5}$	62.7	85.6	**87.7**	64.9	75.22	**4.38**	**7.05**	7.05	70.5
SSR-$\pi_{0.5}$	**71.9**	**88.2**	87.4	**76.5**	**80.99**	4.53	7.90	**6.84**	**67.4**

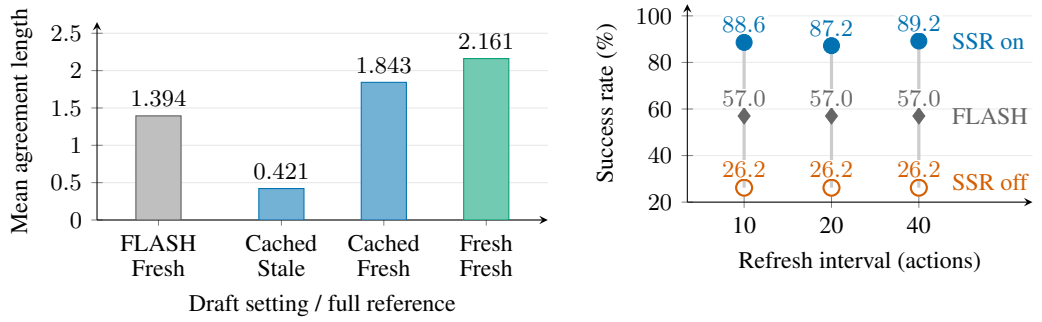
LIBERO-Plus. Standard LIBERO leaves limited room to distinguish task quality, so we further evaluate the success–cost trade-off under the scene and instruction perturbations of LIBERO-Plus (Fei et al., 2026). SSR uses one-step recovery with whole-chunk acceptance, and both methods use an Action verification threshold of 0.15. SSR improves mean SR from 75.22% to 80.99% over FLASH, while mean inference time increases only from 4.38 to 4.53 ms/action. The largest gains occur on Goal and Long. Per-suite results are given in Appendix B.9.

5.3 Analysis of State Recycling

Table 3: Component ablations on LIBERO. All metrics are four-suite means.

Variant	Routing	Rec.	SR (%) ↑	ms/act. ↓	Requests (%) ↓		Full act. (%) ↓
					Prefill	Full	
FLASH	Action + PF2	–	95.85	3.52	41.4	41.4	56.2
Memory draft	Action	–	94.90	**2.96**	**26.1**	26.1	35.5
+ PF2	Action + PF2	–	96.15	3.31	39.7	39.7	49.9
+ Risk	Action + Risk	–	**96.75**	3.17	31.8	31.8	41.9
Full SSR	Action + Risk	✓	96.65	3.05	32.8	**18.4**	**24.8**

System-level ablation. Table 3 shows how the components of SSR change the inference path. Under PF2, the memory-equipped drafter achieves higher mean SR and lower full-action usage than FLASH. Replacing PF2 with Context Risk further increases SR while reducing refresh requests. At the selected operating points, proposal recovery reduces full-action usage from 41.9% to 24.8% with nearly unchanged SR.



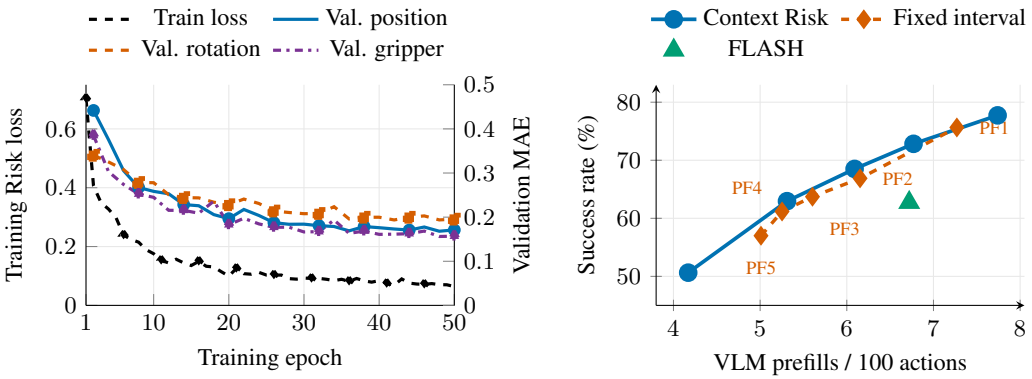
(a) Action agreement **(b)** Draft-only success

Figure 4: **Semantic memory and draft quality on Goal.** (a) Mean agreement-prefix length; labels identify draft memory and policy reference. Cached-memory bars share one draft. (b) Draft-only success with Action memory off/on in the same checkpoint; the SR axis is truncated.

Table 4: One-step recovery from noise or a proposal. Call counts and adoption are four-suite means. Adoption is the fraction of executing at least one action.

Recovery strategy	σ_0	AE steps	Success rate (%) ↑					Calls/100 act.		Adoption (%)
			Goal	Object	Spatial	Long	Mean	Prefill	AE10	
Pure noise	0	1	95.2	99.0	97.4	88.4	95.00	4.70	2.76	81.8
Proposal	0.8	1	**97.2**	**99.4**	**98.6**	**91.4**	**96.65**	**4.44**	2.51	89.8
Proposal	0.9	1	97.0	98.4	97.4	89.2	95.50	4.70	**2.38**	**98.4**

Retained semantics support current prediction. Figure 4a shows that retained memory does not simply reproduce the stale-conditioned policy. Using the same cached-memory draft, the mean offline agreement-prefix length is substantially higher against the fresh policy than against the stale-conditioned policy (1.843 versus 0.421), and refreshing the memory further increases it to 2.161. In draft-only closed-loop execution (Figure 4b), enabling memory yields 87.2–89.2% SR across refresh intervals of 10–40 actions, compared with 26.2% when the memory contribution is removed.



(a) Risk learning on Goal (b) Refresh trade-off on Plus Goal

Figure 5: **Context Risk learning and refresh decisions. (a)** Training loss and validation MAE on Goal. **(b)** Success versus prefills per 100 actions on Goal-Plus; PF1–5 denote fixed refresh intervals.

Context Risk enables context-dependent refresh. Figure 5a shows that the Context Risk target is learnable. More importantly, on LIBERO-Plus Goal, Context Risk achieves a better success–prefill trade-off than fixed periodic refresh at low and intermediate refresh budgets (Figure 5b). Additional routing results are given in Appendix B.7.

Rejected proposals remain useful for recovery. Table 4 compares one-step generation from noise with proposal-initialized recovery under the same AE evaluation budget. At the selected operating points, proposal initialization at $\sigma_0 = 0.8$ reaches 96.65% mean SR versus 95.00% for noise, with recovery adoption of 89.8% versus 81.8% and fewer AE10 calls. The $\sigma_0 = 0.9$ setting has higher adoption (98.4%) but lower mean SR (95.50%), exposing a trade-off between preserving the proposal and correction.

5.4 Real-World Evaluation

Setup. We evaluate task-specific fine-tuned $\pi_{0.5}$ policies with and without SSR on three Franka Panda tasks (Figure 6): conveyor-belt mango grasping, whack-a-mole, and timed four-object pick-and-place. All run on one RTX 4090; the robot holds position during synchronous inference. We use 20 trials per condition for grasping and pick-and-place, and 20 games per speed for whack-a-mole. The 30-second pick-and-place budget includes inference. Appendix C details the settings.

Results. Table 5 shows task-level benefits under time-sensitive synchronous execution. SSR succeeds at conveyor grasping at the highest speed, where full generation fails in all trials, and improves whack-a-mole hit rates at both speeds.

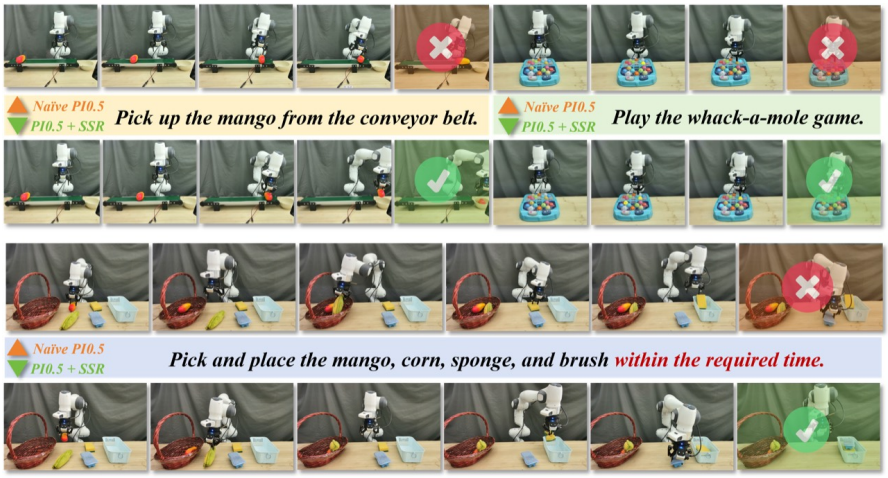


Figure 6: **Real-world execution.** Conveyor grasping (top left), whack-a-mole (top right), and timed four-object pick-and-place (bottom). Each task compares $\pi_{0.5}$ (upper row) with SSR (lower row).

Table 5: Real-world task performance. Success rates for grasping and pick-and-place; pooled hit rates for whack-a-mole (%, 20 trials/games per condition). Speeds denote device presets.

Method	Conveyor-belt grasping			Time-limited pick-and-place	Whack-a-mole	
	High	Mid	Low		High	Low
$\pi_{0.5}$	0.0	35.0	**90.0**	30.0	19.5	55.7
SSR (ours)	**20.0**	**65.0**	**90.0**	**80.0**	**25.8**	**71.3**

Whack-a-mole: 30 seconds/game; hits are pooled over 400 high-speed or 300 low-speed targets across 20 games. Pick-and-place: all four objects must reach their designated containers within 30 seconds of the first post-warmup inference start.

6 Conclusion

Speculative State Recycling (SSR) reuses semantic and action states across conditioning changes to improve Flow-VLA speculative inference. This allows useful speculative computation to remain valuable beyond direct reuse. Simulation results show a favorable success–cost trade-off over FLASH, while real-robot tasks improve over full-generation $\pi_{0.5}$. Asynchronous execution and heterogeneous deployment remain future work.

AI USE STATEMENT

We used generative AI tools to improve the clarity and readability of the manuscript and to assist with organizing and formatting experimental results. The authors take responsibility for the final content of the paper, including all reported results and claims.

REPRODUCIBILITY STATEMENT

Section 4 describes the SSR method, and Appendix A provides prediction and cache implementation details. Section 5 describes the evaluation setup and metrics; Appendix B.4 gives training settings and selected configurations. Appendices B.1 and B.5 specify the inference cost boundary, repeated RTX 4090 timing measurements, and cost aggregation procedure.

REFERENCES

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 *Conference on Robot Learning (CoRL)*, 2025a.

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 *Robotics: Science and Systems (RSS)*, 2025b.

Kevin Black, Manuel Galliker, and Sergey Levine. Real-Time Execution of Action Chunking Flow Policies. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2025c.

Kevin Black, Allen Z. Ren, Michael Equi, and Sergey Levine. Training-Time Action Conditioning for Efficient Real-Time Chunking, 2025d. arXiv preprint arXiv:2512.05964.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. In *International Conference on Machine Learning (ICML)*, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating Large Language Model Decoding with Speculative Sampling, 2023. arXiv preprint arXiv:2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block Diffusion for Flash Speculative Decoding. In *International Conference on Machine Learning (ICML)*, 2026a.

Yang Chen, Xiaoguang Ma, and Bin Zhao. Mean-Flow based One-Step Vision-Language-Action, 2026b. arXiv preprint arXiv:2603.01469.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. DSpark: Confidence-Scheduled Speculative Decoding with Semi-Autoregressive Generation, 2026. arXiv preprint arXiv:2607.05147.

Valentin De Bortoli, Alexandre Galashov, Arthur Gretton, and Arnaud Doucet. Accelerated Diffusion Models via Speculative Sampling. In *International Conference on Machine Learning (ICML)*, 2025.

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: A Progressive Robustness Benchmark for Visual-Language-Action Models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One Step Diffusion via Shortcut Models. In *International Conference on Learning Representations (ICLR)*, 2025.

Gemma Team. Gemma: Open Models Based on Gemini Research and Technology, 2024. arXiv preprint arXiv:2403.08295.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, Zico Kolter, and Kaiming He. Mean Flows for One-step Generative Modeling. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & Faster Large Language Models via Multi-token Prediction. In *International Conference on Machine Learning (ICML)*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations (ICLR)*, 2022.

Hengyuan Hu, Aniket Das, Dorsa Sadigh, and Nima Anari. Diffusion Models are Secretly Exchangeable: Parallelizing DDPMs via Auto Speculation. In *International Conference on Machine Learning (ICML)*, 2025.

Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 1964.

Wenqi Jiang, Jason Clemons, Karu Sankaralingam, and Christos Kozyrakis. How Fast Can I Run My VLA? Demystifying VLA Inference Performance with VLA-Perf, 2026. arXiv preprint arXiv:2602.18397.

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 *Conference on Robot Learning (CoRL)*, 2025.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. In *International Conference on Machine Learning (ICML)*, 2023.

Ye Li, Yuan Meng, Zewen Sun, Kangye Ji, Chen Tang, Jiajun Fan, Xinzhu Ma, Shu-Tao Xia, Zhi Wang, and Wenwu Zhu. SP-VLA: A Joint Model Scheduling and Token Pruning Approach for VLA Model Acceleration. In *International Conference on Learning Representations (ICLR)*, 2026a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. In *International Conference on Machine Learning (ICML)*, 2024.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Zekai Li, Jiaming Tang, and Zhijian Liu. FlashVLA: Streaming Action Decoding for Fast and Asynchronous VLA Inference. In *Conference on Robot Learning (CoRL)*, 2026b. To appear.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow Matching for Generative Modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2023.

Zheng Liu, Zeyu Guo, Zihan Liu, Anbang Wu, Han Zhao, Fangxin Liu, Zhezhi He, Yinhe Han, Jingwen Leng, Minyi Guo, Yiming Gan, and Yu Feng. Deltoris: Enabling Real-time VLA Inference in Embodied AI via Bit-level Sparsity and Speculative Inference. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2026. To appear.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running VLAs at Real-time Speed, 2025. arXiv preprint arXiv:2510.26742.

Jiahui Niu, Kefan Gu, Yucheng Zhao, Shengwen Liang, Tiancai Wang, Xing Hu, Ying Wang, and Huawei Li. Realtime-VLA FLASH: Speculative Inference Framework for Diffusion-based VLAs, 2026. arXiv preprint arXiv:2605.13778.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

Chunyu Qi, Zhuoran Song, Jian Weng, Haozhe Jiang, Xueyuan Liu, Naifeng Jing, Guanghui He, Xiaoyao Liang, and Haibing Guan. SpecVLA: Enabling Efficient Robotic Manipulation via Speculative VLA Inference and Verification. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2026. To appear.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomputing*, 2024.

Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. VLASH: Real-Time VLAs via Future-State-Aware Asynchronous Inference. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026. To appear.

Philippe Tillet, H. T. Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (MAPL)*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

Hanzhen Wang, Jiaming Xu, Yushun Xiang, Jiayi Pan, Yongkang Zhou, Yong-Lu Li, and Guohao Dai. SpecPrune-VLA: Accelerating Vision-Language-Action Models via Action-Aware Self-Speculative Pruning. In *International Conference on Machine Learning (ICML)*, 2026.

Songsheng Wang, Rucheng Yu, Zhihang Yuan, Chao Yu, Feng Gao, Yu Wang, and Derek F. Wong. Spec-VLA: Speculative Decoding for Vision-Language-Action Models with Relaxed Acceptance. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025a.

Zhendong Wang, Max Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, Ming-Yu Liu, and Yu Zeng. One-Step Diffusion Policy: Fast Visuomotor Policies via Diffusion Distillation. In *International Conference on Machine Learning (ICML)*, 2025b.

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. arXiv preprint arXiv:2602.00686.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, Yaxin Peng, Feifei Feng, and Jian Tang. TinyVLA: Toward Fast, Data-Efficient Vision-Language-Action Models for Robotic Manipulation. *IEEE Robotics and Automation Letters*, 2025.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. VLA-Cache: Efficient Vision-Language-Action Manipulation via Adaptive Token Caching. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

Yantai Yang, Yuhao Wang, Zichen Wen, Luo Zhongwei, Chang Zou, Zhipeng Zhang, Chuan Wen, and Linfeng Zhang. EfficientVLA: Training-Free Acceleration and Compression for Vision-Language-Action Models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2025.

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. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft & Verify: Lossless Large Language Model Acceleration via Self-Speculative Decoding. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Zihao Zheng, Zhihao Mao, Maoliang Li, Jiayu Chen, Xinhao Sun, Zhaobo Zhang, Donggang Cao, Hong Mei, and Xiang Chen. KERV: Kinematic-Rectified Speculative Decoding for Embodied VLA Models. In *ACM/IEEE Design Automation Conference (DAC)*, 2026a. To appear.

Zihao Zheng, Zhihao Mao, Sicheng Tian, Maoliang Li, Jiayu Chen, Xinhao Sun, Zhaobo Zhang, Xuanzhe Liu, Donggang Cao, Hong Mei, and Xiang Chen. HeiSD: Hybrid Speculative Decoding for Embodied Vision-Language-Action Models with Kinematic Awareness, 2026b. arXiv preprint arXiv:2603.17573.

A Additional Method Details

This section provides architectural and execution details omitted from Section 4 for clarity.

A.1 Memory-Conditioned Action and Risk Prediction

Each Action and Risk query former uses normalized cross-attention to the retained VLM features H_τ, followed by a residual feed-forward block and output normalization. The two branches use independent memory-reader queries, query-former parameters, decoder queries, channel-wise scales, and output heads.

For the Risk branch, the memory-conditioned queries are constructed analogously to the Action queries in Eq. (4):

$$Z^R_\tau = Q_R(U^R, H_\tau), \qquad \tilde{R}_h = R_h + \gamma_R \odot Z^R_{\tau,h}, \tag{11}$$

where $U^R \in \mathbb{R}^{H\times d}$ are learned memory-reader queries, $R_h \in \mathbb{R}^d$ are Risk decoder queries, and $\gamma_R \in \mathbb{R}^d$ is a learned channel-wise scale. We use rotary position embeddings (Su et al., 2024). Paired Action and Risk queries use the same horizon-relative position because they refer to the same action index.

Both memory projections are computed only when the VLM context is refreshed. Subsequent speculative rounds reuse Z^A_τ and Z^R_τ while updating the current visual-language and robot-state inputs. Thus, a single shallow forward produces both Action and Risk predictions without rerunning either query former on every speculative round.

The attention mask keeps the two predictions separate within this shared computation. Action queries may attend to the current inputs and Action queries, while Risk queries may attend to the current inputs and Risk queries; the two query blocks cannot attend to each other. Current input tokens do not attend to either query block.

A.2 Why Condition Decoder Queries with Memory?

The addition in Eq. (4) conditions the decoder query before it reads the current inputs. The learned vector A_h supplies a base query for action position h, while the query former supplies a context-dependent residual in the same d-dimensional space:

$$\delta_h = \Gamma_A Z^A_{\tau,h}, \qquad \Gamma_A = \mathrm{diag}(\gamma_A), \qquad \tilde{A}_h = A_h + \delta_h. \tag{12}$$

The residual preserves the query dimension and adds no memory tokens to the shallow block's input sequence. Setting $\delta_h = 0$ recovers the base query in the same model. The channel-wise scale is a globally learned parameter initialized to small nonzero values, rather than a gate predicted from each observation.

To see how this residual affects observation reading, consider one attention head of G. Let ψ_h map a decoder embedding to its attention query, including the head's normalization, projection, and positional transformation. Define the base query and its memory-induced change as

$$q^0_h = \psi_h(A_h), \qquad \Delta q_h = \psi_h(A_h + \delta_h) - \psi_h(A_h), \qquad q_h = q^0_h + \Delta q_h. \tag{13}$$

This decomposition is exact even when ψ_h includes nonlinear normalization; it does not assume that $\Delta q_h = \psi_h(\delta_h)$. For two current-input tokens i, j visible to this query, let $k_{t,i}, k_{t,j} \in \mathbb{R}^{d_k}$ be their attention keys and $p_{h,i}, p_{h,j}$ their softmax attention weights. These current-input keys do not depend on H_τ, because the current inputs do not read the memory-conditioned query blocks. For scaled dot-product attention (Vaswani et al., 2017), their relative weights satisfy

$$\begin{aligned}\log \frac{p_{h,i}}{p_{h,j}} &= \frac{q_h^\top (k_{t,i} - k_{t,j})}{\sqrt{d_k}} \\ &= \frac{(q^0_h)^\top (k_{t,i} - k_{t,j})}{\sqrt{d_k}} + \frac{(\Delta q_h)^\top (k_{t,i} - k_{t,j})}{\sqrt{d_k}}.\end{aligned} \tag{14}$$

The second term is a memory-dependent change in the preference between current-input tokens. Thus, retained features can influence which parts of the latest observation the shallow predictor

reads, even though memory extraction occurs only at refresh. The identity concerns relative attention to current inputs; the block can also attend to its own branch queries and applies its usual residual and feed-forward transformations.

By comparison, simply appending memory tokens to fixed current-input tokens in one attention layer leaves the original query and current-input keys unchanged. Those memory tokens can change the total attention mass assigned to current inputs, but do not change their pairwise attention ratios through that query. Conditioning the query first introduces the interaction in Eq. (14) within the single shallow block. This explains the role of pre-attention memory injection; it does not establish that additive fusion is uniquely optimal or guarantee improved action quality. The same reasoning applies to the independently parameterized Risk queries.

A.3 Training Objectives and Context Routing

The Action branch learns to reproduce the fresh full-policy actions $A_t^\star = A_t^{\text{fresh}}$ with the per-action prediction loss ℓ_{act}:

$$\mathcal{L}_A = \frac{1}{H}\sum_{h=1}^{H} \ell_{\text{act}}(\hat{A}_{t,h}, A_{t,h}^\star). \tag{15}$$

The Risk branch is trained with

$$\mathcal{L}_R = \frac{1}{H}\sum_{h=1}^{H}\sum_{c\in\mathcal{C}} \text{Huber}\left(\hat{D}_{t,h,c} - D_{t,h,c}\right), \tag{16}$$

where $\mathcal{C} = \{\text{pos}, \text{rot}, \text{grip}\}$. Training uses the weighted objective $\lambda_R\mathcal{L}_R$ for Risk-specific parameters.

Let θ_G denote the shared shallow-block parameters and θ_A, θ_R the Action- and Risk-specific parameters. Gradients are routed as

$$g_{G,A} = \nabla_{(\theta_G,\theta_A)}\mathcal{L}_A, \qquad g_R = \nabla_{\theta_R}(\lambda_R\mathcal{L}_R). \tag{17}$$

Risk gradients pass through the shared block to reach Risk-specific inputs but do not update θ_G or θ_A. The shared block and Action branch are therefore optimized only by the Action objective.

Each training frame is paired with a plausible historical cache from the same trajectory. Each current training frame contributes one Action target and at most one stale–fresh pair per epoch, so historical-cache sampling does not multiply Action supervision. The Risk labels compare stale- and fresh-conditioned full-policy outputs using the same current state and flow noise, and therefore remain independent of the lightweight proposal.

For inference, each discrepancy channel is normalized by a fixed scale η_c derived from held-out validation discrepancies. The runtime threshold λ is selected independently of the continuous training target and remains fixed across inference rounds. Changing λ adjusts the tolerance for reusing cached context and therefore trades refresh frequency against executable draft yield.

A.4 Proposal Recovery and Cost Accounting

In Eq. (9), the same flow time σ_0 determines both the proposal–noise interpolation and the starting point of subsequent AE integration. This keeps the initialization consistent with the flow parameterization: larger σ_0 preserves more proposal structure and leaves a shorter fresh-conditioned integration interval. The step count k controls the number of sequential AE evaluations over the remaining interval.

Recovery always uses fresh conditioning K_f and robot state s_f obtained from the same observation. The recovery noise is sampled independently of the verification noise; the saved proposal provides initialization structure rather than a previously integrated state of the fresh-policy trajectory.

Every fresh VLM pass updates the AE conditioning and both semantic memories. A saved proposal is consumed once and invalidated after use or reset. If recovery fails its displacement check, full AE generation reuses the same fresh KV cache and memory projections without another VLM prefill or query-former pass.

For completeness, recovery adds a third possible source of executed actions to the draft and full-generation paths defined in Section 3. Let N_R denote actions executed from recovered prefixes and T_R the complete inference time of logical rounds that produce them. Then

$$\mathcal{E}_{\mathrm{SSR}} = \frac{N_D + N_R + N_F}{T_D + T_R + T_F} = \frac{1}{\bar{T}_{\mathrm{action}}}. \tag{18}$$

Each logical round is assigned according to its final action source, and all preceding speculative, refresh, recovery, and fallback computation is included in its complete cost. In particular, a failed recovery followed by full generation includes the recovery attempt but does not count a second VLM prefill because the fresh conditioning is reused.

B Supplementary Details

B.1 Complete Inference Cost Accounting

The path costs in Eqs. (2) and (18) account for all inference work in logical rounds, grouped by the final action source. Simulation costs follow the timing and aggregation protocol in Appendix B.5. A logical round may span multiple serving requests; all associated inference work is attributed to its eventual output path. The cost boundary includes image encoding, memory processing, model computation, and internal routing overhead; it excludes observation acquisition, RPC, external action denormalization, and physical action execution. Initialization and scheduled full rounds include only the operations they execute, without assuming a preceding draft or verification.

Failed recovery followed by full generation contributes both the recovery attempt and full-generation cost to T_F, without counting a second prefill when fresh KV is reused. Inference-usage and executed-action statistics cover the evaluated episodes, including failed tasks and timeouts; zero-output requests add cost without adding actions. Efficiency is a ratio of totals, not an average of round-level ratios. We report success rate alongside mean model time per executed action.

B.2 Gripper Guard and Context Risk

The gripper guard and Context Risk address different decisions. Risk predicts how the full policy's actions change when stale VLM conditioning is replaced by fresh conditioning, including the gripper component. A small predicted difference does not rule out a gripper transition: both policies can predict the same transition. It therefore does not determine whether the gripper rule should schedule full generation.

SSR preserves the original gripper verification, prefix truncation, and scheduled full request. Gripper-triggered refresh takes priority over Risk and Action rejection and runs fresh VLM conditioning followed by AE10, without proposal recovery. The recovery displacement bound checks continuous pose and excludes the gripper, so passing it does not replace the gripper guard either.

B.3 Per-Suite Inference Cost and Usage

Table 6 reports the per-suite breakdown corresponding to Table 1. SSR reduces mean inference time relative to FLASH on all four suites under the compiled backend.

Table 6: Success and inference cost by suite. Runs match Table 1. Torch denotes torch.compile; calls are per 100 executed actions.

Suite	Method	SR (%) ↑	Mean ms/action	Calls/100 act.		Full (%)	
				Prefill	AE10	Req.	Act.
Goal	FLASH Torch	97.2	3.33	5.22	5.22	40.3	52.2
	SSR Torch	97.2	3.16	**4.90**	**2.44**	17.9	**23.8**
	SSR Triton	**97.4**	**2.45**	5.03	2.54	**17.8**	23.9
Object	FLASH Torch	**99.6**	3.88	6.50	6.50	45.7	65.0
	SSR Torch	99.4	2.84	**3.62**	**2.57**	**19.2**	**25.7**
	SSR Triton	98.2	**2.11**	3.75	2.67	19.3	26.7
Spatial	FLASH Torch	98.0	3.69	5.89	5.89	41.6	58.8
	SSR Torch	**98.6**	3.26	**4.41**	**2.89**	19.2	**28.9**
	SSR Triton	97.0	**2.50**	4.66	3.00	**19.1**	30.0
Long	FLASH Torch	88.6	3.20	4.87	4.87	38.0	48.7
	SSR Torch	**91.4**	2.93	**4.81**	**2.14**	17.4	**20.6**
	SSR Triton	91.2	**2.22**	**4.81**	2.15	**17.2**	20.8

B.4 Training and Evaluation Settings

We train a separate Action–Risk head for each suite on NVIDIA RTX 4090 GPUs using teacher action targets and historical-cache pairs. Checkpoints are selected by execution-prefix action error on held-out validation episodes. Training uses a per-device micro-batch of 32 with two accumulation steps and learning rates 2×10^{-4} for Action/shared parameters and 3×10^{-4} for Risk parameters. The Risk loss weight is 0.2. Risk normalization uses the 90th percentile of per-position validation discrepancies; the runtime threshold λ is varied separately. Training cache gaps extend to 40, with gaps $\{6, 11, 15, 22, 30, 38\}$ reserved for validation.

The Risk threshold λ controls the tolerance for reusing cached context and is adjusted to trade task success against refresh frequency.

Table 7: Training episodes and selected checkpoints.

Suite	Train episodes	Validation episodes	Selected epoch
Goal	386	42	47
Object	409	45	45
Spatial	390	42	47
Long	342	37	45

B.5 Inference Environment and Timing

To reduce the variability of individual timing measurements, we benchmark each inference component separately on an RTX 4090 (24 GB), using 8 warmup iterations followed by 40 timed repetitions, and report the mean. We use PyTorch (Paszke et al., 2019) 2.7.1+cu126 and Triton (Tillet et al., 2019) 3.3.1. Inputs contain two 224×224 views, 512 visual tokens, and 200 valid language tokens, with an action horizon of 10. The compiled backend uses torch.compile; the Triton backend also uses CUDA graphs and cuBLAS.

SSR runs both query formers during the initial full pass and each subsequent VLM refresh, then reuses their cached outputs on speculative rounds. The joint draft forward includes both Action and Risk prediction. If recovery fails, full generation reuses the fresh KV cache without another prefill or query-former pass.

Table 8 reports the component means to show the distribution of inference cost. We also calibrate complete execution paths on the same GPU to account for routing and state-update overhead in per-action costs. Two-request paths include both requests; the gripper-triggered full path excludes the preceding trigger request.

Table 8: Mean component time on RTX 4090 (ms). Wall-time means over 40 repetitions after 8 warmups. Comp. denotes torch.compile.

Component	FLASH		SSR	
	Comp.	Triton	Comp.	Triton
Image encoding	7.110	4.584	7.232	4.740
VLM refresh[a]	20.258	19.990	22.630	22.387
Query formers (included)	–	–	1.557	1.376
Draft prediction[b]	2.699	0.596	2.732	0.814
Action verification	3.695	1.665	3.677	1.684
Recovery AE step[c]	–	–	2.839	1.289
Full AE generation (10 steps)	17.719	11.128	17.646	11.120

[a] SSR refresh includes both query formers; their combined time is also measured separately and must not be added again.
[b] SSR draft prediction jointly includes Action and Risk.
[c] One recovery AE step excludes the outer displacement check, prefix truncation, and state update. FLASH does not use proposal recovery.

For recorded or reconstructed path counts n_i, calibrated mean path times $\bar{t}_i$, and N_{act} executed actions, mean model time per action is

$$\widehat{C}_{\text{act}} = \frac{\sum_i n_i \bar{t}_i}{N_{\text{act}}}. \tag{19}$$

Timing includes image encoding and model computation, excluding observation acquisition, RPC, external action denormalization, and environment execution.

B.6 Memory-Scale Intervention

We vary the Action-memory residual scale α on Goal while keeping the trained checkpoint fixed. Here $\alpha = 1$ retains the learned memory contribution and $\alpha = 0$ removes it at inference; neither setting retrains the drafter. Risk, periodic refresh, and recovery are disabled throughout, so $\alpha = 1$ corresponds to the memory-draft configuration rather than full SSR. Scaling memory also leaves its refresh schedule unchanged: full strength does not make a cached representation fresh.

Table 9 shows that memory improves offline agreement with fresh full-policy actions: the mean endpoint prefix increases from 1.634 at $\alpha = 0$ to 1.843 at $\alpha = 1$. The offline test uses 46,548 paired validation states without gripper filtering, as in Figure 4. The closed-loop SR difference is smaller, at 94.6% versus 95.2% (three additional successful episodes). Intermediate scales do not improve SR, even when offline agreement is slightly higher. These results support retaining the learned memory contribution, while showing that offline agreement alone does not determine closed-loop success.

B.7 Risk Routing and Threshold Sensitivity

With recovery disabled, Table 10 compares Context Risk, PF2, and neither Risk nor periodic refresh. Risk uses fewer prefills than PF2 on Goal, Object, and Spatial, and more on Long at these operating points. Risk thresholds match the subsequent recovery experiments on Object, Spatial, and Long;

Table 9: Action-memory scaling on Goal. A fixed checkpoint is evaluated over 500 episodes per setting. Offline prefixes measure agreement with fresh full-policy actions.

α	Offline	Closed-loop evaluation			
	Endpoint prefix	SR (%) ↑	Executed prefix	Full actions (%)	Prefill/100
1.00	1.843	**95.2**	7.41	35.7	3.57
0.75	**1.880**	93.4	7.42	35.6	3.56
0.50	1.857	92.6	7.37	34.8	3.48
0.25	1.766	88.2	7.44	34.0	3.40
0.00	1.634	94.6	7.28	36.4	3.64

Table 10: Routing with recovery disabled. Rows share a checkpoint within each suite, with 500 episodes per setting. Prefill equals AE10 calls per 100 actions.

Suite	Routing	SR (%) ↑	Prefill/100 ↓	Full actions (%)
Goal	PF2	**97.6**	4.97	49.7
	No Risk, PF0	95.2	3.57	35.7
	Static Risk	97.2	4.15	41.5
Object	PF2	**99.2**	5.16	51.6
	No Risk, PF0	98.2	3.66	36.6
	Static Risk	99.0	3.57	35.7
Spatial	PF2	97.4	5.45	54.5
	No Risk, PF0	96.8	4.38	43.8
	Static Risk	**98.0**	4.35	43.5
Long	PF2	90.4	4.37	43.7
	No Risk, PF0	89.4	2.58	25.8
	Static Risk	**92.8**	4.70	47.0

Goal uses $\lambda = 1.2$ without recovery and $\lambda = 1.0$ for the selected full SSR setting. The Long PF2 entry uses the same run as Table 3.

We fix recovery to $\sigma_0 = 0.8$, $k = 1$, and prefix bound $\rho = 0.15$ on Goal. With recovery fixed on Goal, increasing the Risk tolerance reduces prefills per 100 actions from 6.10 to 3.89, while SR changes from 97.4% to 96.0% (Table 11). SR varies non-monotonically over the tested thresholds.

Table 11: Risk tolerance on Goal. A–D increase the tolerance with recovery fixed; each point uses 500 episodes. Adoption is conditional on recovery attempts.

Operating point	SR (%) ↑	Prefill/100 ↓	AE10/100 ↓	Adoption (%)	Mean ms/act. ↓
A	**97.4**	6.10	2.70	84.7	–
B	97.2	4.90	2.44	83.1	3.16
C	93.6	4.15	2.38	85.2	–
D	96.0	**3.89**	**2.34**	84.8	–

Table 12: Whole-chunk and prefix recovery on LIBERO. All settings use one AE step and $\rho = 0.15$, with 500 episodes per row.

Suite	Rule	σ_0	SR (%) ↑	Calls / 100 actions		Adoption (%)
				Prefill	AE10	
Goal	Whole	0.8	96.4	5.96	3.69	57.2
	Prefix	0.8	**97.2**	4.90	2.44	83.1
	Prefix	0.9	97.0	6.04	2.28	98.1
Object	Whole	0.8	97.8	3.71	2.84	76.8
	Prefix	0.8	**99.4**	3.62	2.57	95.1
	Prefix	0.9	98.4	3.64	2.52	99.6
Spatial	Whole	0.8	98.0	4.43	3.59	46.5
	Prefix	0.8	**98.6**	4.41	2.89	86.8
	Prefix	0.9	97.4	4.41	2.69	97.3
Long	Whole	0.8	**92.0**	4.78	2.51	80.4
	Prefix	0.8	91.4	4.81	2.14	94.2
	Prefix	0.9	89.2	4.72	2.04	98.7

B.8 Recovery Bounds and Starting Times

Table 12 compares whole-chunk acceptance with two prefix-recovery starting times at the selected Risk thresholds. Goal uses $\lambda = 1.0$ for prefix recovery at $\sigma_0 = 0.8$ and $\lambda = 0.8$ for the other two settings. At $\sigma_0 = 0.8$, prefix acceptance reduces AE10 calls on Object and Long; SR changes from 97.8% to 99.4% on Object and from 92.0% to 91.4% on Long. On Long, increasing σ_0 to 0.9 raises recovery adoption to 98.7%, while SR falls to 89.2%.

B.9 LIBERO-Plus Per-Suite Results

On LIBERO-Plus, SSR uses one recovery step with $\sigma_0 = 0.8$ and requires every action in the recovered chunk to satisfy the displacement bound $\rho = 0.15$. The original gripper truncation still applies after acceptance.

Table 13 gives per-suite success rates, inference usage, and mean model time for FLASH and SSR. Full-action and full-round shares use executed actions and inference rounds as their respective denominators. SSR uses fewer AE10 calls than FLASH on Goal, Spatial, and Long, and more on Object; prefill counts are higher on all four suites.

Table 13: Per-suite results on LIBERO-Plus. Settings match Table 2. Time uses the compiled backend; calls are per 100 executed actions.

Suite	Method	SR (%) ↑	ms/act. ↓	Full actions (%) ↓	Full rounds (%) ↓	Prefill/100 ↓	AE10/100 ↓
Goal	FLASH	62.70	**4.18**	67.20	**42.20**	**6.72**	6.72
	SSR	**71.90**	4.49	**66.60**	46.20	6.83	**6.68**
Object	FLASH	85.60	**4.21**	**67.00**	**41.40**	**6.70**	**6.70**
	SSR	**88.20**	4.74	73.20	60.50	8.53	7.44
Spatial	FLASH	**87.70**	5.34	89.10	**45.50**	**8.91**	8.91
	SSR	87.40	**5.13**	**87.60**	70.30	9.15	**8.79**
Long	FLASH	64.90	3.79	58.60	**39.20**	**5.86**	5.86
	SSR	**76.50**	**3.74**	**42.30**	39.90	7.10	**4.47**

Model time per action uses the same path timings and weighting procedure as Table 1. SSR weights use recorded call statistics; the Goal calculation combines the new call densities with historical direct-full and recovery-trigger proportions. FLASH weights use AE10 density and the full-round fraction. SSR's mean time is lower than FLASH on Spatial and Long, and higher on Goal and Object.

C Real-World Deployment

Task setups. All three tasks use a shared tabletop workspace, with the scene and robot reset between trials. For conveyor-belt grasping, a tabletop conveyor carries a mango, and the robot must grasp it and place it into a receiving container. We evaluate 20 trials per method at each of the low, medium, and high preset speeds; all other outcomes count as failures. For whack-a-mole, a game device is placed in front of the robot, which presses illuminated buttons using its closed gripper. Each game lasts 30 seconds and presents 15 targets at low speed or 20 targets at high speed. We evaluate 20 games per method at each speed, yielding 300 and 400 target appearances, respectively. Successful presses are recorded by the device counter and checked from recorded videos. The reported hit rate is the total number of successful presses divided by the total number of target appearances across all games. For time-limited pick-and-place, a mango, corn, sponge, and brush are placed between containers on the left and right sides of the workspace. Each method is evaluated over 20 trials. Success requires all four objects to be in their designated containers within 30 seconds, timed from the start of the first policy inference after warmup.

Real-world platform. Figure 7 shows the robot platform and task props. We use a Franka Emika Panda equipped with two cameras: a front-mounted Orbbec Femto Bolt and an Intel RealSense D405 mounted at the gripper flange. These provide front and wrist observations for all three tasks. The shared platform accommodates the tabletop conveyor, whack-a-mole device, manipulation objects, and receiving containers.

Training and inference. We collect 50 demonstrations per task and independently fine-tune three $\pi_{0.5}$ policies using LoRA (Hu et al., 2022) in openpi[1] on eight NVIDIA RTX 4090 GPUs. Conveyor-belt and whack-a-mole demonstrations are collected at the devices' low-speed settings. Each pick-and-place demonstration contains the complete four-object sequence. We train the SSR Action–Risk predictor for the corresponding task following Section 4. Table 14 summarizes the base-policy fine-tuning and SSR training settings. Both use the AdamW optimizer (Loshchilov & Hutter, 2019). All policies run on a single NVIDIA RTX 4090, with the baseline $\pi_{0.5}$ using Triton inference. Both methods use synchronous execution: the robot holds its position during inference and executes the returned action prefix before requesting the next prediction.

[1] https://github.com/Physical-Intelligence/openpi

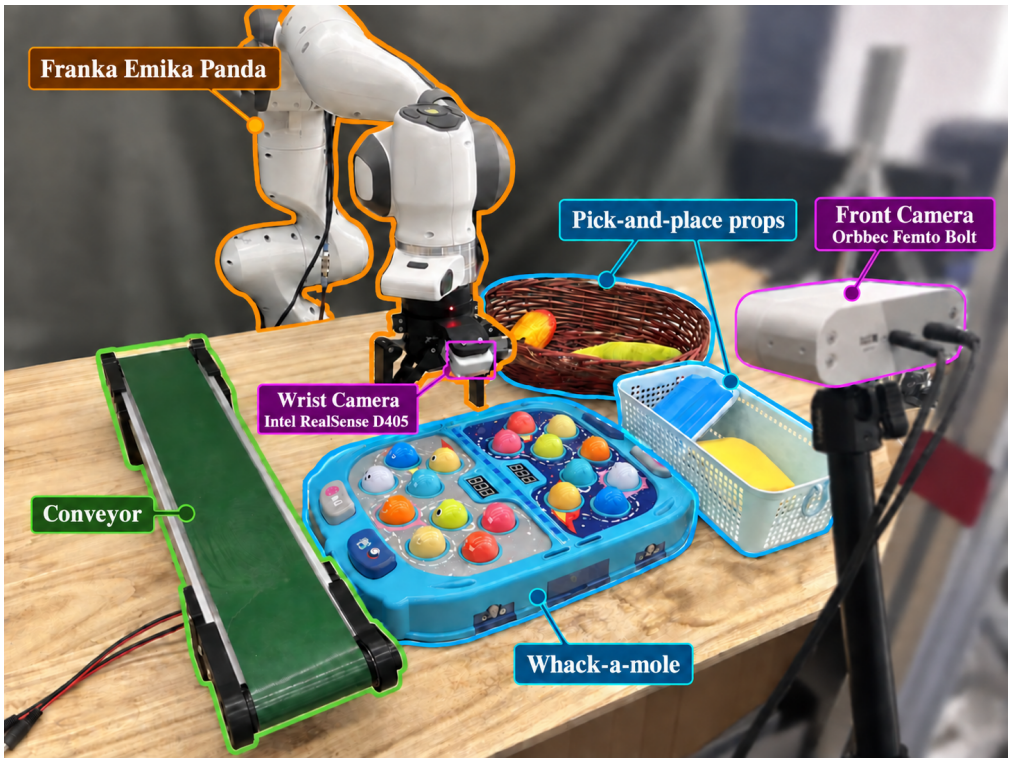


Figure 7: **Real-world experimental platform.** Annotated illustration of the Panda robot, camera arrangement, and props used for conveyor-belt mango grasping, whack-a-mole, and time-limited four-object pick-and-place. The platform uses an Orbbec Femto Bolt for the front view and a flange-mounted RealSense D405 for the wrist view.

Table 14: Training settings for real-world adaptation. A separate $\pi_{0.5}$ policy is fine-tuned for each task. Action parameters include the shared backbone and Action-specific modules; Risk parameters are specific to the Risk branch.

Parameter	Value
Base policy fine-tuning	
Pretrained checkpoint	$\pi_{0.5}$ base
Adaptation	LoRA
Demonstrations	50 per task
Training hardware	$8\times$ NVIDIA RTX 4090
FSDP devices	8
Training steps / global batch size	10,000 / 8
Optimizer	AdamW
Adam (β_1, β_2) / ϵ	$(0.9, 0.95)$ / 10^{-8}
Weight decay	10^{-10}
Learning-rate schedule	Cosine decay with 1,000 warmup steps
Peak learning rate	2.5×10^{-5}
Schedule duration	30,000 steps
Schedule endpoint learning rate	2.5×10^{-6}
Gradient clipping	Global norm 1.0
EMA decay	0.99
SSR Action–Risk training	
$\pi_{0.5}$ backbone / draft precision	BF16 / FP32
Batch size / gradient accumulation	16 / 1
Optimizer	AdamW
Adam (β_1, β_2) / ϵ	$(0.9, 0.999)$ / 10^{-8}
Weight decay	0.01
Action / Risk peak learning rate	1.0×10^{-4} / 1.5×10^{-4}
Learning-rate schedule	Cosine decay with 2 warmup epochs
Minimum learning rate	10% of each peak learning rate
Gradient clipping	Global norm 1.0
Maximum training epochs	50
Validation interval	Every 2 epochs
Minimum epochs before early stopping	20
Early-stopping patience	8 consecutive validation checks without improvement

Base-policy training stops after 10,000 steps while retaining the default 30,000-step learning-rate schedule; the schedule endpoint learning rate is therefore not reached during training.