Realtime-VLA FLASH: A Speculative Inference Framework for Diffusion-based VLAs

Jiahui Niu[1,2] **Kefan Gu**[3,4] **Yucheng Zhao**[4] **Shengwen Liang**[1*]
Tiancai Wang[4*] **Xing Hu**[1] **Ying Wang**[1] **Huawei Li**[1]
[1] State Key Lab of Processors, Institute of Computing Technology, CAS
[2] University of Chinese Academy of Sciences
[3] Nanjing University [4] Dexmal
Page: `https://dexmal.github.io/realtime-vla-flash`

Abstract

Diffusion-based vision-language-action models (dVLAs) are promising for embodied intelligence but are fundamentally limited in real-time deployment by the high latency of full inference. We propose **Realtime-VLA FLASH**, a speculative inference framework that eliminates most full inference calls during replanning by introducing a lightweight draft model with parallel verification via the main model's Action Expert and a phase-aware fallback mechanism that reverts to the full inference pipeline when needed. This design enables low-latency, high-frequency replanning without sacrificing reliability. Experiments show that on LIBERO, FLASH largely preserves task performance by replacing many 58.0 ms full-inference rounds with speculative rounds as fast as **7.8 ms**, lowering task-level average inference latency to 19.1 ms (**3.04**× speedup). We additionally demonstrate effectiveness on real-world conveyor-belt sorting, highlighting its practical impact for latency-critical embodied tasks.

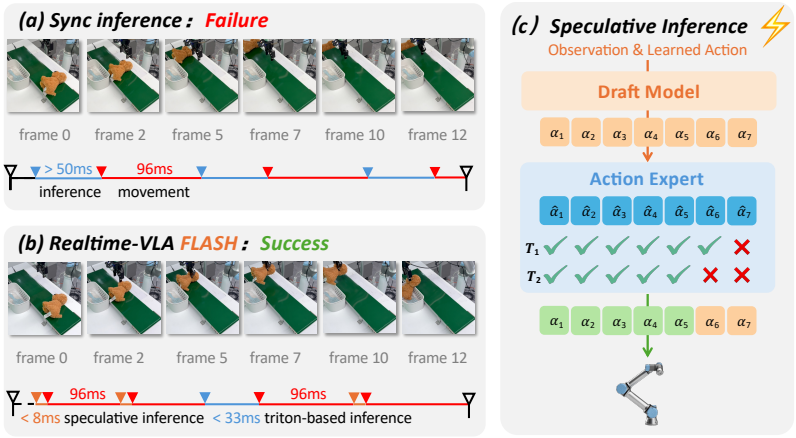


Figure 1: **Overview of Realtime-VLA FLASH.** (a) Standard synchronous dVLA inference runs the full pipeline at each replanning round, causing stale action updates and failure in latency-critical manipulation. (b) Realtime-VLA FLASH accelerates replanning, allowing the robot to react in time and complete the grasp. (c) FLASH introduces speculative inference for dVLAs: a lightweight draft model proposes a continuous action chunk, and the main model's Action Expert reconstructs action endpoints from selected flow-matching timesteps in parallel to check consistency with the draft. The robot executes the longest consistent prefix and falls back to the full path when verification fails.

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

1 Introduction

Diffusion-based vision-language-action models (dVLAs) [1, 10, 20, 21] have recently emerged as an important paradigm for embodied intelligence. Their diffusion-based continuous action generation captures multimodal action distributions, making them well suited to complex manipulation tasks.

However, deploying dVLAs on real robotic systems remains challenging. In practice, robot control typically runs at a much higher frequency than model inference, as shown in Figure 1, and existing systems therefore rely on open-loop action chunking [33] to bridge this mismatch. For example, π_0 [1] predicts a chunk of future actions and replans only after several control steps have already been executed. While this design reduces the frequency of replanning, each replanning round still requires the same expensive full inference pipeline. As a result, end-to-end inference latency remains the main bottleneck, limiting the applicability of dVLAs to reactive, latency-sensitive tasks.

Speculative Inference aims to reduce repeated inference cost and has proven effective in large language models [12–14] and autoregressive VLAs [26, 34, 35]. Its core idea is to have a lightweight draft model propose candidates that are verified in parallel by the main model, thereby reducing expensive full generations. However, extending this paradigm to dVLAs is non-trivial. Speculative inference requires (1) a draft proposal, (2) parallel verification, and (3) an acceptance criterion, which are all well-defined for autoregressive models through token-level probabilities. dVLAs instead produce actions through iterative denoising in continuous space [5, 16], which challenges each ingredient: it is unclear what constitutes a meaningful draft through multi-step denoising, the sequential nature of denoising resists parallel verification, and the absence of explicit likelihoods leaves no natural acceptance criterion. The central bottleneck is therefore verification: *without a way to cheaply check a drafted continuous-action chunk, speculative inference collapses back to full sequential denoising.*

A key observation is that flow matching provides exactly such a structure. During flow-matching training, interpolation timesteps are sampled along linear paths between Gaussian noise and target actions, learning local flow constraints at intermediate states. Given a drafted action chunk, we can interpolate it with Gaussian noise, evaluate a few intermediate states in parallel using the Action Expert, and check whether the reconstructed endpoints agree with the draft, providing a cheap consistency check without full sequential denoising. In addition, consistency verification does not make speculative inference uniformly safe throughout the trajectory. For much of task execution, the robot performs smooth motions, where observations change slowly and small draft errors can often be tolerated. In fine-adjustment phases such as gripper switching, these errors can quickly accumulate and be amplified, leading to task failure. This motivates falling back to the full inference pipeline when verification or the task phase indicates that higher-fidelity actions are required.

In this paper, we introduce **Realtime-VLA FLASH**, a speculative inference framework for dVLAs. FLASH introduces a fast speculative path that uses a lightweight draft model to quickly propose a candidate action chunk, and then verifies its consistency with the main model's Action Expert in parallel at a few selected timesteps, producing a dynamically sized executable action prefix. If no action prefix is accepted, FLASH falls back to the full path to correct the trajectory. FLASH also uses gripper switches as heuristic signals of upcoming fine-adjustment phases, where higher-quality full-path actions help prevent draft errors from accumulating into task failures. Thus, FLASH reduces average inference latency and improves action throughput while largely preserving task success.

We evaluate Realtime-VLA FLASH in simulation and real-world tasks. On LIBERO, with Triton optimization, FLASH runs speculative rounds as fast as **7.8 ms**, compared with 58.0 ms for full-inference rounds, lowering the task-level average latency to 19.1 ms (**3.04**$\times$ speedup) and improving action throughput by **2.63**$\times$, while largely preserving task performance. On real conveyor-belt sorting, FLASH enables successful grasping at belt speeds up to 15 m/min, where other methods fail, demonstrating that speculative inference reduces effective inference latency and enables higher-speed, latency-sensitive manipulation tasks.

In summary, we make three contributions. First, we formulate speculative inference for flow-matching dVLAs and address its key challenge, continuous-action verification, using flow-matching interpolation paths. Second, we introduce Realtime-VLA FLASH, a dual-path inference runtime that combines a lightweight draft model, flow-consistency-based parallel verification, and phase-aware fallback to reduce expensive full-path inference during replanning. Third, we demonstrate on LIBERO and real-world tasks that FLASH replaces many expensive full-path rounds with low-cost speculative rounds, substantially reducing latency while largely preserving task success.

2 Related Work

2.1 Efficient Diffusion-based VLAs

Existing work has explored several ways to improve the efficiency of dVLAs [1, 10, 20, 21], among which the following two directions are most closely related to our setting.

Inference Pipeline Acceleration. Existing work on efficient dVLAs primarily reduces the cost of the original full inference pipeline through smaller models [28, 22, 15], layer compression [31], token pruning [25], quantization [28, 29, 32], and kernel/system-level optimization [19, 30]. These approaches improve the efficiency of the original inference pipeline, either by reducing the cost of particular components or by improving end-to-end execution efficiency. Our work is complementary to this line of research, but it addresses a different question at the control-loop level: whether every replanning round needs to invoke the full path.

Diffusion and Flow-Matching Acceleration. Another line of work accelerates diffusion- or flow-based policies [5, 16] by shortening the generation process itself, for example through distillation into few-step models [18, 27, 7] or direct one-step model training [8, 4]. These approaches reduce the cost of sequential denoising more directly, but they typically require additional training or modified policy formulations. In contrast, we keep the original flow-matching formulation intact and improve efficiency by exploiting otherwise idle compute during the Action Denoise stage.

2.2 Speculative Inference

Speculative Inference accelerates autoregressive generation by letting a lightweight draft model propose candidate outputs and a larger model verify them in parallel, thereby reducing expensive decoding steps. It has been widely studied in large language models [12–14] and extended to autoregressive VLAs [26, 34, 35], where discrete action tokens admit token-level probabilities that naturally support verification and acceptance.

Related acceleration has also been explored for stochastic diffusion models, where drafted denoising states can be checked using the transition structure of the stochastic reverse process [6, 9]. However, π_0 [1]-style dVLAs differ from both settings: they generate continuous action chunks under flow matching by integrating an ODE-defined velocity field, not discrete tokens or samples from a stochastic reverse kernel. Consequently, neither token-level verification nor transition-based diffusion verification applies directly, and the absence of explicit likelihoods leaves no natural acceptance criterion for drafted continuous-action chunks.

3 Methods

In this section, we present the architecture of Realtime-VLA FLASH. Section 3.1 revisits the dVLA inference pipeline and the latency bottlenecks revealed by profiling; Section 3.2 then summarizes the overall two-path design, and Sections 3.3–3.5 present the three core components of the flash path, namely draft action generation, multi-step parallel verification, and phase-aware fallback.

3.1 dVLA Preliminaries and Latency Analysis

dVLAs, such as π_0 [1], typically combine a pre-trained vision-language model (VLM) backbone [23, 24] with an Action Expert (AE). Perceptual context is passed from the VLM to the AE through the KV Cache. Formally, the model takes a multimodal observation

$$o_t = [I_t^1, \ldots, I_t^n, \ell_t, q_t],$$

where I_t^i denotes the images, ℓ_t the language prompt, and q_t the robot state, and predicts a future action chunk

$$A_t = [a_t, a_{t+1}, \ldots, a_{t+H-1}].$$

Each per-step action a_t contains semantically distinct channels corresponding to end-effector position Δp_t, rotation $\Delta\theta_t$, and binary gripper control g_t. Under chunked execution, only a prefix of A_t is executed before the next action chunk is regenerated from the updated observation; we refer to each such refresh as a replanning round.

At inference time, the input images are first encoded into visual features, then concatenated with the language features, and the resulting multimodal prefix is processed by the VLM to construct a prefix KV Cache. This cache is then passed to the Action Expert, which predicts the action chunk by solving an observation-conditioned ODE from Gaussian noise:

$$\frac{dA_t^\tau}{d\tau} = v_\theta(A_t^\tau, \tau, o_t), \qquad A_t^0 \sim \mathcal{N}(0, I),$$

where A_t^τ denotes the intermediate noisy action state at denoising time τ. At each denoising step, the Action Expert predicts the instantaneous denoising velocity field $v_\theta(A_t^\tau, \tau, o_t)$, and the final action chunk is obtained by numerically integrating this learned field over multiple timesteps, typically 10 denoising steps in the default setting, until $\tau = 1$. Consequently, a full inference round in π_0-style dVLAs consists of three stages: Image Encoder, VLM prefill to construct the visual KV Cache, and a multi-step Action Denoise stage under the learned flow field.

We further profile these three stages on an NVIDIA RTX 4090D [11] and summarize the results with a roofline analysis in Figure 2. The measured latencies of Image Encoder, VLM prefill, and Action Denoise are 11.3 ms, 26.7 ms, and 20.0 ms, respectively. The roofline places Image Encoder and VLM prefill primarily in the compute-bound regime, suggesting that their latency should be reduced by eliminating redundant computation. For VLM prefill, this naturally motivates reusing the previously computed prefix KV Cache. In contrast, Action Denoise is memory-bound, since each denoising step repeatedly reads the cache while remaining unable to parallelize across steps. This leaves available compute resources underutilized and motivates parallel verification instead of repeatedly invoking the full path.

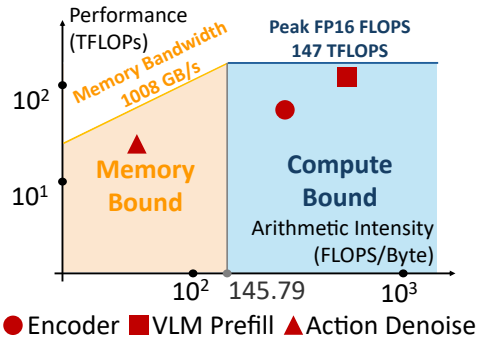


Figure 2: **Roofline analysis** of π_0 [1] inference on NVIDIA RTX 4090D. 145.79 is the balanced arithmetic intensity point.

3.2 Realtime-VLA FLASH Overview

Motivated by Section 3.1, Realtime-VLA FLASH adopts the dual-path architecture shown in Figure 3(a). In a flash-path round, FLASH still encodes the current images and feeds the features to a lightweight draft model, which proposes a candidate action chunk. The main model's Action Expert then reuses the visual KV Cache produced by the most recent full-path round to reconstruct action endpoints at selected denoising timesteps and check their consistency. If no drafted action prefix

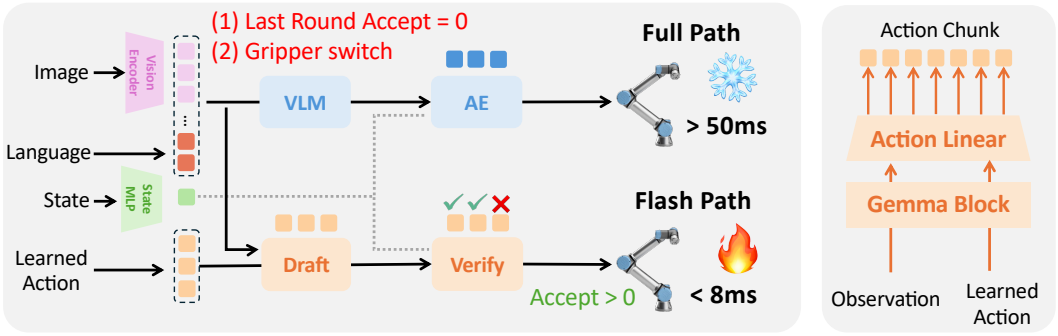


Figure 3: **Realtime-VLA FLASH Framework.** (a) Realtime-VLA FLASH uses two inference paths: the original full path (Full Path) and a lightweight speculative path (Flash Path). The Full Path performs Image Encoder, VLM prefill, and Action Denoise, while the flash path still encodes the current images but skips VLM prefill to draft a candidate action chunk for verification. (b) Draft model architecture. The draft model uses a single Gemma block with the same structure as the VLM, augments the input with learned action queries, and predicts the full action chunk in parallel through a linear action head.

is accepted, the cached context may no longer provide a reliable basis for speculative execution; FLASH therefore returns to the full path to refresh the context and correct the trajectory. In addition, FLASH uses gripper switches as phase-transition signals and proactively falls back to the full path during fine-adjustment phases, where higher-fidelity actions help mitigate error amplification.

3.3 Draft Action Generation

For flash-path rounds, candidate generation must be cheaper than invoking the full path. We therefore design a lightweight draft model that proposes a candidate action chunk for subsequent verification.

The draft model architecture in Figure 3(b) consists of a single Gemma block followed by a linear action head. At inference time, the flash path still runs the image encoder of the main model on the current images and combines the resulting visual features with the language instruction, while the robot state is projected into the same hidden space through a linear layer. We then append H learned action queries to the input sequence, one for each future timestep in the chunk. The attention mask follows the blockwise structure of the base model: visual and language tokens form the conditional prefix block, the state token forms a separate block, and the action queries form the action block. The action block attends to the full conditional prefix, allowing the model to predict the entire candidate chunk in parallel. The draft model is trained by supervised regression to the target action chunk,

$$\mathcal{L}_{\text{draft}} = \sum_{h=1}^{H} w_h \, \ell\big(\hat{a}^{d}_{t+h-1},\ a_{t+h-1}\big)\,,$$

where w_h is a step-dependent weight and $\ell(\cdot,\cdot)$ is the per-step action regression loss.

This design has several advantages. The draft model is lightweight, containing about 110M parameters in our implementation, compared with roughly 2.7B parameters in the VLM, which makes candidate generation substantially cheaper than invoking the full path. It also remains architecturally close to the main VLM and preserves the same action-chunk representation, making it easy to integrate and allowing the verification stage to reuse the Action Expert. Additional architectural and training details are provided in Appendix A.

3.4 Multi-step Parallel Verification

The remaining challenge is to verify a draft action chunk without rerunning the full sequential denoising process in Figure 4(a). In flow matching, this becomes possible because the policy learns the velocity field along the linear path between Gaussian noise and the endpoint at training time.

For a draft endpoint $\hat{A}_t^{(d)}$ and Gaussian noise ϵ, we construct an intermediate denoising state as

$$\tilde{A}_\tau = \tau \hat{A}_t^{(d)} + (1-\tau)\epsilon,$$

where τ denotes denoising progress from Gaussian noise to the action endpoint. Under this convention, the Action Expert predicts the local velocity from the intermediate state toward the endpoint. Since the remaining denoising interval is $1-\tau$, we reconstruct the endpoint by

$$\hat{A}_t(\tau) = \tilde{A}_\tau + (1-\tau)\, v_\theta(\tilde{A}_\tau, \tau \mid c_t,\ s_t).$$

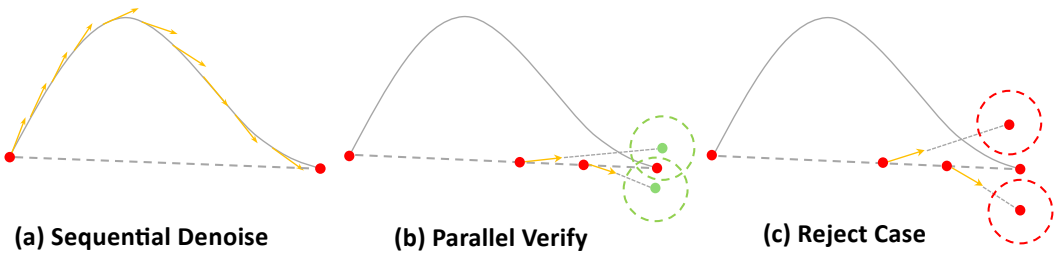


Figure 4: **Parallel verification.** (a) π_0 [1] generates an action chunk with flow matching through 10-step sequential denoising. (b) Realtime-VLA FLASH reconstructs endpoints from selected denoising timesteps in parallel and checks an action-by-action distance threshold within the chunk, yielding the accepted prefix. (c) If reconstructed endpoints deviate beyond the threshold, no prefix is accepted and the flash path falls back to the full path.

Algorithm 1 Multi-Step Parallel Verification

Require: Draft action chunk $\hat{A}_t^{(d)}$, reused visual KV Cache c_t, latest robot state s_t, verification timesteps $\mathcal{T} = \{\tau_1, \ldots, \tau_K\}$, normalized threshold δ_{norm}, numerical floor $\varepsilon > 0$

Ensure: Executable prefix length L

Sample a shared Gaussian noise ϵ for all verification timesteps

for all $\tau_k \in \mathcal{T}$ **in parallel do**

  Construct intermediate state: $\tilde{A}_{\tau_k}^{(k)} = \tau_k \hat{A}_t^{(d)} + (1 - \tau_k)\epsilon$

  Predict local denoising velocity with the Action Expert: $v^{(k)} = v_\theta\left(\tilde{A}_{\tau_k}^{(k)}, \tau_k \,\middle|\, c_t,\ s_t\right)$,

  Reconstruct endpoint estimate: $\hat{A}_t^{(k)} = \tilde{A}_{\tau_k}^{(k)} + (1 - \tau_k)\, v^{(k)}$

  for $h = 1, \ldots, H$ **do**

    Compute normalized step distance: $d_h^{(k)} = \frac{\text{Dist}_{\text{cont}}\left(\hat{a}_{t+h-1}^{(d)}, \hat{a}_{t+h-1}^{(k)}\right)}{\max(1-\tau_k, \varepsilon)}$

  end for

  Compute prefix length: $L^{(k)} = \sum_{h=1}^{H} \prod_{j=1}^{h} \mathbf{1}\left[d_j^{(k)} \le \delta_{\text{norm}}\right]$

end for

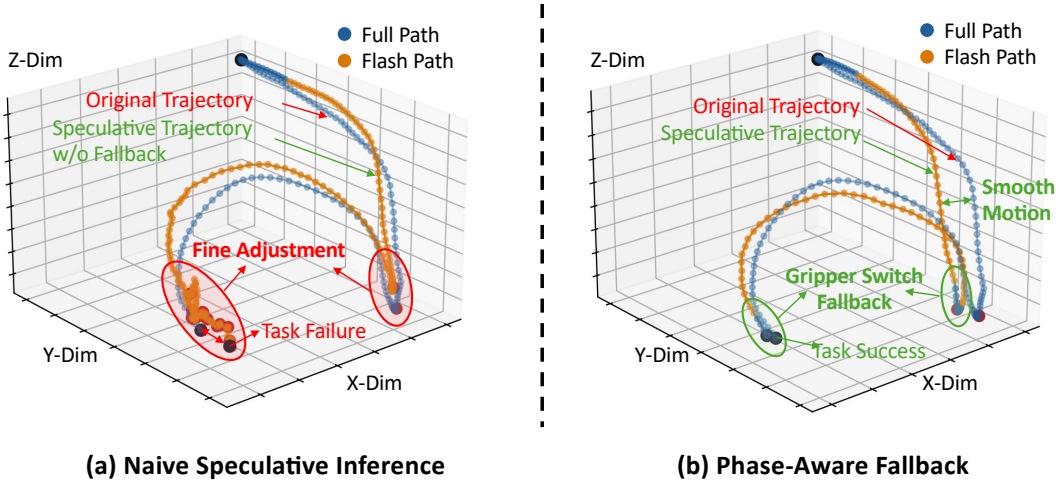
Return the conservative prefix length: $L = \min_{k=1,\ldots,K} L^{(k)}$

If the draft is consistent with the main policy under the reused KV Cache c_t and the latest robot state s_t, these reconstructed endpoints should remain close to the draft across verification timesteps.

We therefore evaluate a small set of verification timesteps $\mathcal{T} = \{\tau_1, \ldots, \tau_K\}$ in parallel and define the executable prefix as the longest leading segment whose included actions all satisfy the distance threshold at every verification timestep, as summarized in Algorithm 1 and Figure 4(b). We apply this distance-based rule to the continuous position and rotation channels through $\text{Dist}_{\text{cont}}$, while the gripper channel is treated separately because it encodes discrete semantic states rather than incremental motion. If $L = 0$, no executable prefix survives all verification checks, so the draft is rejected and control immediately returns to the full path, corresponding to the rejection case in Figure 4(c). Compared with binary acceptance, this rule returns an executable prefix, enabling more fine-grained execution and a smoother handoff to fallback. See Appendix B for further discussion.

3.5 Phase-aware Fallback

Verification in Section 3.4 is local: it checks the current draft under the reused context but does not anticipate transitions into fine-adjustment phases. Figure 5 compares the LIBERO-Spatial bowl-



(a) Naive Speculative Inference **(b) Phase-Aware Fallback**

Figure 5: **Phase-aware fallback on a LIBERO-Spatial task trajectory.** 3D trajectories for a bowl-to-plate task. (a) Without phase-aware fallback, the flash path drifts during final placement and fails near the plate edge. (b) With fallback, final placement returns to the full path and succeeds.

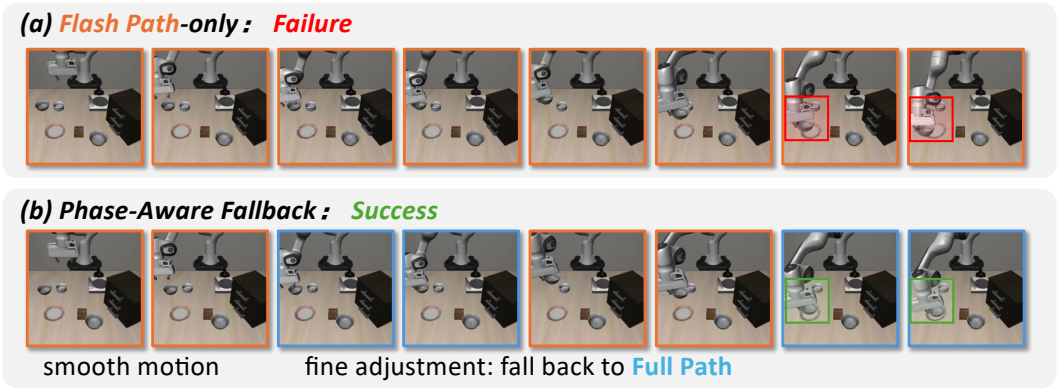


Figure 6: **Key-frame visualization of Figure 5.** Orange borders indicate flash-path execution and blue borders indicate full-path execution after fallback. The failure rollout keeps final placement on the flash path, leaving the bowl misaligned with the plate.

to-plate trajectory without and with phase-aware fallback. Without fallback, flash-path execution continues into the final fine-adjustment phase, where the trajectory drifts toward the plate edge and leaves the bowl misaligned with the plate. Figure 6 provides key frames.

We therefore augment verification-triggered fallback with a phase-aware fallback rule based on the gripper channel, as shown in Figure 5(b). Unlike the continuous position and rotation channels, the gripper channel represents discrete open and closed modes, encoded as -1 and 1 in LIBERO. After applying the same standardization used by the policy, the two modes remain separated around zero, so we detect gripper switches by thresholding the standardized gripper value at zero. Such switches correspond to grasp or release events, which often mark precision-critical fine-adjustment phases.

If a gripper switch appears in any verification branch within the candidate chunk, we treat it as evidence that the rollout has entered a fine-adjustment phase. The system then falls back immediately to the full path to regenerate higher-quality actions, avoiding error accumulation in precision-sensitive phases where small deviations can cause task failure.

4 Experiments

4.1 Setup

We evaluate Realtime-VLA FLASH on π_0 [1] in both simulation and real-world reactive manipulation tasks. All online inference experiments are run on a single NVIDIA RTX 4090D. We use an action chunk size of $H = 50$ and replan after every 12 executed actions.

Simulation. We follow the standard LIBERO setup [17] and report results on four suites: LIBERO-Spatial, LIBERO-Object, LIBERO-Goal, and LIBERO-10. Each suite contains 10 tasks, and each task is evaluated over 50 trials. We train a separate draft model for each LIBERO suite.

Real-world. For real-world experiments, we use a single-arm UR5 platform equipped with two Intel RealSense D435i cameras, which provide multi-view observations of the workspace. We evaluate conveyor-belt sorting with two object categories: toy dog and hairbrush. For each object category, we collect 200 demonstrations and fine-tune the policy with LoRA for 20k steps. Each real-world condition is evaluated over 10 trials.

4.2 Simulation Evaluation

Setup. We evaluate Realtime-VLA FLASH on LIBERO by comparing four methods: Torch-π_0 [1], Triton-π_0 [19], FLASH-π_0, and FLASH+Triton-π_0. We adapt Triton-π_0 to LIBERO and build FLASH+Triton-π_0 by combining our FLASH with the Triton implementation. The structurally aligned draft and verify modules allow partial reuse of existing Triton-π_0 kernels. We report suite-level success rates, average task-level inference latency (**Lat.**), and per-action latency (**/Act**).

Table 1: **Simulation results on four LIBERO suites.** We report suite-level success rates, average success rate (**SR**), average task-level latency (**Lat. (ms)**), average per-action latency (**/Act (ms)**), and relative improvement over the original π_0 [1].

Methods	Success Rate (%)				Average			Improvement	
	Spatial	Object	Goal	10	SR	Lat.	/Act	ΔSR	Speedup
Torch-π_0 [1]	96.8	98.8	**95.8**	85.2	94.1	58.0	5.0	–	1.00×
Triton-π_0 [19]	96.4	98.8	95.0	**86.6**	94.2	39.7	3.5	+0.1	**1.46**×
FLASH-π_0	96.4	99.2	94.6	83.4	93.4	34.9	3.0	-0.7	**1.66**×
FLASH+Triton-π_0	**96.8**	**99.2**	94.4	84.6	93.8	19.1	1.9	-0.3	**3.04**×

Table 2: **Suite-level flash-path statistics on LIBERO. Acc.** is the average accepted draft prefix length in flash-path rounds, normalized by the replan size (12). **Flash Path Rate (FR.)** is the fraction of flash-path rounds among all replanning rounds across task trajectories.

LIBERO Suites	Acc. (%)		FR. (%)		Lat. (ms)		/Act (ms)	
	FLASH	+Tri.	FLASH	+Tri.	FLASH	+Tri.	FLASH	+Tri.
Spatial	75.8	60.2	71.6	68.9	34.3	19.2	3.0	2.1
Object	78.1	**89.3**	**72.4**	**86.4**	34.6	**12.6**	3.0	**1.2**
Goal	81.4	66.9	63.7	62.4	37.5	20.5	3.3	2.1
10	**82.4**	62.5	57.5	49.6	**33.2**	24.1	**2.8**	2.3
Avg.	79.4	69.7	66.3	66.8	34.9	19.1	3.0	1.9

Results. Table 1 summarizes the main simulation results. Compared with the 58.0 ms original full-path round, a flash-path round costs only 17.9 ms for FLASH-π_0 and **7.8 ms** for FLASH+Triton-π_0 (Appendix A, Table 6). With phase-aware fallback, FLASH-π_0 reduces the task-level average latency to 34.9 ms, achieving a 1.66× speedup over Torch-π_0; with Triton optimization, FLASH+Triton-π_0 further reduces it to **19.1 ms**, achieving a **3.04**× speedup. This also corresponds to a **2.63**× reduction in per-action latency, with only a 0.3-point drop in average success rate.

Table 2 attributes the FLASH speedup to high flash-path usage: FLASH+Triton-π_0 handles 66.8% of replanning rounds via the flash path, with accepted prefixes covering 69.7% of the replan window, thereby reducing full-path calls and amortizing speculative execution. Table 3 evaluates runtime components on LIBERO-10, the most hyperparameter-sensitive suite, where small speculative errors can accumulate and cause task failure. We fix the verification threshold to $\delta = 0.15$ and use two verifier timesteps ($\mathcal{T} = \{0.90, 0.95\}$), then ablate periodic full-path refresh (**PF**) and phase-aware fallback (**FB**). PF periodically invokes higher-fidelity full-path actions to correct long-horizon drift, while FB performs targeted correction near precision-critical phases indicated by gripper switches. We finally select +FB & PF=2 as the best success-latency trade-off for the result in Table 1, with the ablations provided in Appendix C.1.

Table 3: **Ablation of key components on LIBERO-10.** PF = n denotes forcing one full-path refresh every n flash-path rounds.

Configuration	SR (%)	Lat. (ms)
Baseline (Flash-path only)	58.4	**13.3**
Periodic full-path refresh		
+ PF = 2	80.6	21.0
+ PF = 3	82.0	18.6
+ PF = 4	75.4	17.2
Phase-aware fallback		
+ FB	66.8	17.7
+ FB & PF = 2	**84.6**	24.1
+ FB & PF = 3	81.6	21.9
+ FB & PF = 4	78.4	20.2

4.3 Real-world Evaluation

Setup. We evaluate real-world conveyor-belt sorting, where the robot must grasp a moving object and finish sorting before it leaves the reachable region, as shown in Figure 7. To isolate policy inference latency, all methods use the same synchronous inference loop without asynchronous inference, real-time chunking [2, 3], or other latency-hiding optimizations. We compare JAX-π_0 [1], Triton-π_0 [19], and FLASH+Triton-π_0 under the same hardware and protocol across medium, high, and extra high speeds. Appendix C.2 provides the platform and training hyperparameters.

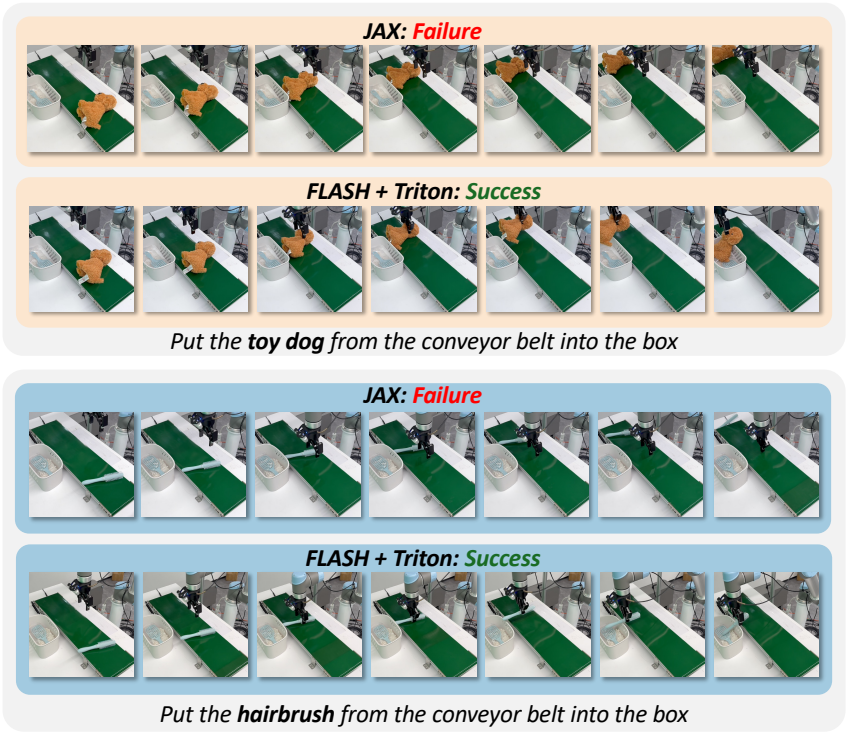


Figure 7: Conveyor-belt sorting demo.

Table 4: **Conveyor-belt sorting results under synchronous inference.** We report success rate (%) over 10 trials for each object and speed; demonstrations are collected at 6 m/min.

Methods	Medium (10 m/min)		High (13 m/min)		Extra High (15 m/min)	
	Toy dog	Hairbrush	Toy dog	Hairbrush	Toy dog	Hairbrush
JAX-π_0 [1]	20.0	50.0	0.0	0.0	0.0	0.0
Triton-π_0 [19]	80.0	40.0	30.0	10.0	0.0	0.0
FLASH+Triton-π_0	**80.0**	**90.0**	**50.0**	**30.0**	**20.0**	**10.0**

Results. Table 4 shows that conveyor speed amplifies the effect of policy inference latency. As a reference point below the reported speed range, JAX-π_0 achieves 80% success on toy dog at 8 m/min, but drops sharply at the reported speeds. At medium speed, all methods achieve nonzero success, but FLASH+Triton-π_0 already improves the more timing-sensitive hairbrush task. As speed increases, the synchronous baselines degrade rapidly: JAX-π_0 [1] fails at high speed, Triton-π_0 [19] retains only limited success, and both fail at extra high speed. In contrast, FLASH+Triton-π_0 remains successful at extra high speed and is the only method with nonzero success at 15 m/min.

Failures of the slower baselines mostly come from stale action chunks: the robot approaches an outdated belt position, so the gripper arrives behind the object or closes too late. This effect is stronger at higher speeds and on hairbrushes, whose thin geometry leaves less tolerance to timing and pose errors. These failures support the main real-world takeaway: under synchronous control, reducing policy latency directly expands the speed range in which dVLAs can complete reactive manipulation.

5 Conclusion

We presented Realtime-VLA FLASH, a speculative inference framework that serves most replanning rounds through the flash path and reverts to the full path when verification or fine-adjustment phases require higher fidelity. Experiments on LIBERO and conveyor-belt sorting show substantial latency reductions while largely preserving task performance, suggesting a practical path toward latency-sensitive embodied control. FLASH still relies on heuristic thresholds, leaving trajectory-adaptive verification thresholds as an important direction for future work.

Acknowledgements

We thank Haoqiang Fan and collaborators for insightful discussions and for the Realtime-VLA line of work [19, 30], which helped shape our focus on latency-aware VLA deployment. This work is supported in part by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDB0660103, the National Natural Science Foundation of China (NSFC) under Grant No. 62202453, the Beijing Municipal Science and Technology Program under Grant No. Z241100004224012, and the Beijing Natural Science Foundation (4262032, L2607007).

References

[1] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *ArXiv*, abs/2410.24164, 2024.

[2] Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*, 2025.

[3] Kevin Black, Allen Z Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025.

[4] Yang Chen, Xiaoguang Ma, and Bin Zhao. Mean-flow based one-step vision-language-action. *arXiv preprint arXiv:2603.01469*, 2026.

[5] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[6] Valentin De Bortoli, Alexandre Galashov, Arthur Gretton, and Arnaud Doucet. Accelerated diffusion models via speculative sampling. *arXiv preprint arXiv:2501.05370*, 2025.

[7] Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. *arXiv preprint arXiv:2410.12557*, 2024.

[8] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. *arXiv preprint arXiv:2505.13447*, 2025.

[9] Hengyuan Hu, Aniket Das, Dorsa Sadigh, and Nima Anari. Diffusion models are secretly exchangeable: Parallelizing ddpms via autospeculation. *arXiv preprint arXiv:2505.03983*, 2025.

[10] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael 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, 2025. URL `https://arxiv.org/abs/2504.16054`.

[11] Wenqi Jiang, Jason Clemons, Karu Sankaralingam, and Christos Kozyrakis. How fast can i run my vla? demystifying vla inference performance with vla-perf. *arXiv preprint arXiv:2602.18397*, 2026.

[12] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

[13] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 7421–7432, 2024.

[14] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

[15] Tao Lin, Yilei Zhong, Yuxin Du, Jingjing Zhang, Jiting Liu, Yinxinyu Chen, Encheng Gu, Ziyan Liu, Hongyi Cai, Yanwen Zou, et al. Evo-1: Lightweight vision-language-action model with preserved semantic alignment. *arXiv preprint arXiv:2511.04555*, 2025.

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

[17] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

[18] Songming Liu, Bangguo Li, Kai Ma, Lingxuan Wu, Hengkai Tan, Xiao Ouyang, Hang Su, and Jun Zhu. Rdt2: Exploring the scaling limit of umi data towards zero-shot cross-embodiment generalization. *arXiv preprint arXiv:2602.03310*, 2026.

[19] Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running vlas at real-time speed. *arXiv preprint arXiv:2510.26742*, 2025.

[20] NVIDIA, Johan Bjorck, Nikita Cherniadev Fernando Castañeda, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots. In *ArXiv Preprint*, March 2025.

[21] NVIDIA GEAR Team, Allison Azzolini, Johan Bjorck, Valts Blukis, et al. Gr00t n1.6: An improved open foundation model for generalist humanoid robots. `https://research.nvidia.com/labs/gear/gr00t-n1_6/`, December 2025.

[22] 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.

[23] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[24] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[25] Hanzhen Wang, Jiaming Xu, Yushun Xiang, Jiayi Pan, Yongkang Zhou, Yong-Lu Li, and Guohao Dai. Specprune-vla: Accelerating vision-language-action models via action-aware self-speculative pruning. *arXiv preprint arXiv:2509.05614*, 2025.

[26] 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 *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 26916–26928, 2025.

[27] Zhendong Wang, Zhaoshuo Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, et al. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. *arXiv preprint arXiv:2410.21257*, 2024.

[28] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

[29] Justin Williams, Kishor Datta Gupta, Roy George, and Mrinmoy Sarkar. Lite vla: Efficient vision-language-action control on cpu-bound edge robots. *arXiv preprint arXiv:2511.05642*, 2025.

[30] Chen Yang, Yucheng Hu, Yunchao Ma, Yunhuan Yang, Jing Tan, and Haoqiang Fan. Realtime-vla v2: Learning to run vlas fast, smooth, and accurate. *arXiv preprint arXiv:2603.26360*, 2026.

[31] 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. *arXiv preprint arXiv:2506.10100*, 2025.

[32] Jingxuan Zhang, Yunta Hsieh, Zhongwei Wang, Haokun Lin, Xin Wang, Ziqi Wang, Yingtie Lei, and Mi Zhang. Quantvla: Scale-calibrated post-training quantization for vision-language-action models. *arXiv preprint arXiv:2602.20309*, 2026.

[33] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[34] 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. *arXiv preprint arXiv:2603.01581*, 2026.

[35] Zihao Zheng, Zhihao Mao, Sicheng Tian, Maoliang Li, Jiayu Chen, Xinhao Sun, Zhaobo Zhang, Xuanzhe Liu, Donggang Cao, Hong Mei, et al. Heisd: Hybrid speculative decoding for embodied vision-language-action models with kinematic awareness. *arXiv preprint arXiv:2603.17573*, 2026.

A Draft Model Details

A.1 Architecture

This section expands the draft architecture introduced in Section 3.3. Because the draft model reuses a VLM block, the observations provide visual context but do not define where a fixed-length action chunk should be written. We therefore append H learned action slots as explicit output positions:

$$Q = [q_1, q_2, \ldots, q_H],$$

where q_h corresponds to the h-th future action. These slots are trainable parameters, and they serve as persistent output queries that decode structured future actions from the shared context.

We use a blockwise attention mask to make these slots compatible with the reused VLM architecture. Each action slot attends to the visual-language prefix and state token, and the slots can also interact within the action block, allowing the predicted actions to be coordinated across the chunk. Because all H slots are processed in the same forward pass, the transformer produces hidden states for the entire action chunk in parallel.

After the transformer block, only the hidden states associated with the action slots are decoded. A shared linear action head maps each slot representation to its corresponding continuous action:

$$\hat{a}^{(d)}_{t+h-1} = W_{\mathrm{act}} z_h + b_{\mathrm{act}}, \qquad h = 1, \ldots, H,$$

where z_h is the final hidden representation of slot q_h. Thus, the learned action slots convert a VLM block into a parallel action-chunk predictor.

A.2 Training Hyperparameters and Inference Cost Breakdown

Table 5 summarizes the draft-model training setup. The draft model regresses to frozen full-path policy outputs with a prefix-weighted objective, so early executable actions receive higher weight while verification and fallback still decide the final executed prefix.

Table 5: Training hyperparameters for the draft model.

Item	Value
Optimizer	AdamW
Learning rate	2×10^{-3}
Weight decay	0.01
Batch size	64
Epochs	100
Precision	FP32
Loss	Weighted Smooth L1 / Huber loss, $\beta = 1.0$
Prefix weighting	Sampled prefix up to 16 steps, $\gamma_{\mathrm{prefix}} = 0.9$, tail weight 0.1
Checkpoint	Best validation RMS over the first 12 executable steps
Compute cost	4 NVIDIA RTX 4090D GPUs, approximately 6 hours per draft model

Table 6 reports the component-level latency behind the task-level results in Table 1. We include both per-round component costs and trajectory-level averages because flash-path rounds may execute variable-length prefixes, so the trajectory-level latency depends on how often each path is used rather than only on the per-round costs.

Table 6: Inference cost breakdown on a single NVIDIA RTX 4090D.

Component	Torch-π_0	Triton-π_0	FLASH-π_0	FLASH+Triton-π_0
Full-path round (ms)				
Image encoder	11.3	4.7	11.3	4.7
VLM prefill	26.7	22.4	26.7	22.4
Action denoise	20.0	12.6	20.0	12.6
Full-path total	58.0	39.7	58.0	39.7
Flash-path round (ms)				
Image encoder	–	–	11.0	4.7
Draft model	–	–	3.5	0.9
Parallel verifier	–	–	3.4	2.2
Flash-path total	–	–	**17.9**	**7.8**
Trajectory-level average				
Task-level latency	58.0	39.7	34.9	19.1
Per-action latency	5.0	3.5	3.0	1.9
Speedup	1.00×	1.46×	1.66×	**3.04**×

B Verification Consistency Interpretation

The endpoint check in Algorithm 1 should be interpreted as a heuristic local consistency test, rather than as a formal guarantee that the accepted draft and the full-path rollout induce identical trajectories or task-level behavior. The key distinction is that the Action Expert is trained by flow matching on interpolation paths toward target action endpoints, whereas verification probes the learned flow field on states induced by a drafted endpoint. Thus, passing verification indicates that the draft is locally compatible with the learned velocity field under the reused conditioning context, but it does not establish full-path equivalence.

Because the step distance in Algorithm 1 is divided by $\max(1-\tau_k, \varepsilon)$, an accepted action satisfies $\mathrm{Dist}_{\mathrm{cont}}(\hat{a}^{(d)}_{t+h-1}, \hat{a}^{(k)}_{t+h-1}) \le \delta_k$ at every verification timestep, where $\delta_k = \max(1-\tau_k, \varepsilon)\,\delta_{\mathrm{norm}}$ is the effective endpoint tolerance. Without this normalization, the endpoint residual would shrink with the factor $(1-\tau_k)$, so a fixed raw threshold would be increasingly lenient for later probes.

Let $a^{\star}_{t+h-1}$ denote the action produced by a full-path rollout under the current conditioning context, and let $\hat{a}^{(d)}_{t+h-1}$ denote an accepted action. The per-action discrepancy can be viewed as controlled by the effective tolerance, the local reconstruction error of the Action Expert, the residual conditioning mismatch, and the mismatch between target-induced and draft-induced interpolation paths:

$$\mathrm{Dist}_{\mathrm{cont}}\left(\hat{a}^{(d)}_{t+h-1}, a^{\star}_{t+h-1}\right) \lesssim \max_k \delta_k + \eta_{\mathrm{AE}} + \eta_{\mathrm{cond}} + \eta_{\mathrm{path}}.$$

Here, η_{AE} accounts for approximation errors in the local velocity field and the single-step endpoint reconstruction performed by the Action Expert. The term η_{cond} captures residual mismatch introduced by reusing the cached visual KV from the most recent full-path round; it does not include the robot state, since s_t is refreshed at every verification call (Algorithm 1). The term η_{path} captures the fact that verification evaluates the flow field on draft-induced interpolation states, rather than on the target-endpoint paths used during training or the states visited by a full sequential rollout. This decomposition is a qualitative description; we do not state the assumptions under which it holds as a bound, and the individual terms are not estimated separately. Appendix D.1 instead evaluates the verifier and the effect of the cached visual KV empirically.

Overall, the acceptance rule should be understood as a conservative local agreement criterion, not as a formal correctness guarantee for the accepted prefix. This motivates conservative thresholds, multi-timestep verification, and the fallback mechanisms of the runtime: full-path fallback when no prefix is accepted, phase-aware fallback at gripper switches, and periodic full-path refresh.

C Experimental Details

C.1 Simulation Details: Verification Hyperparameter Ablation

In this subsection, we isolate the verifier by disabling phase-aware fallback and periodic full-path refresh. The results should therefore be read as a verifier trade-off study, rather than as the performance of the complete runtime. Throughout, $\tau = 0$ denotes Gaussian noise and $\tau = 1$ the action endpoint, and δ denotes the threshold δ_{norm} in Algorithm 1.

Selecting the number of verification timesteps. We first study the trade-off associated with the number of verification timesteps, $K = |\mathcal{T}|$. Table 7 reports a joint ablation over K and the distance threshold δ on LIBERO-10, which we use as a verifier stress test.At $\delta = 0.15$, $K = 1$ achieves low latency but accepts nearly the entire replan window on average (99.4% normalized prefix length), providing little filtering. $K = 4$ accepts much shorter prefixes (6.8%) and improves verifier-only success, but its latency approaches the full-path regime. $K = 2$ provides an intermediate operating point with a normalized mean accepted-prefix length of 65.6% at a per-round verifier cost of 2.2 ms. We therefore use $K = 2$ in the subsequent timestep-location and threshold analyses.

Selecting verification timestep locations. With $K = 2$, we next examine how the locations of the verification timesteps affect speculative coverage and task performance. We fix $\delta = 0.15$ and keep both phase-aware and periodic fallback disabled. Because step distances are normalized by $\max(1-\tau_k, \varepsilon)$ in Algorithm 1, the same δ can be applied to all probe locations. Table 8 compares early, intermediate, late, and cross-stage probes on LIBERO-10.

Table 7: Joint verifier ablation over K and δ on LIBERO-10 using the Triton backend, with phase-aware fallback and periodic refresh disabled.

Verifier Setting			LIBERO-10 Result				
K	Verify (ms)	δ	SR (%)	Lat. (ms)	/Act (ms)	Acc. (%)	FR. (%)
4	4.3	0.20	82.6	35.7	4.2	13.9	18.8
		0.15	86.2	39.9	4.4	6.8	9.2
		0.10	85.0	47.5	4.5	1.8	2.1
		0.05	84.4	52.4	4.5	0.0	0.0
		0.00	86.4	52.3	4.5	0.0	0.0
2	2.2	0.20	54.3	10.3	0.9	91.8	94.3
		0.15	**58.4**	**13.3**	**1.5**	**65.6**	**83.0**
		0.10	79.4	21.1	3.3	23.3	40.9
		0.05	87.2	44.9	4.5	2.0	2.5
		0.00	85.6	47.9	4.1	0.0	0.0
1	1.4	0.20	54.9	13.5	1.2	99.7	95.4
		0.15	53.8	12.5	1.1	99.4	95.4
		0.10	51.0	12.9	1.2	84.8	92.4
		0.05	84.2	28.5	4.0	11.4	19.6
		0.00	86.8	49.8	4.2	0.0	0.0

Early and intermediate probes achieve higher success rates but accept very few speculative actions, resulting in substantially higher latency. In contrast, late probes at $\mathcal{T} = \{0.90, 0.95\}$ yield much higher acceptance and lower latency, but reduce success when used without fallback. We therefore select late probes to preserve useful speculative coverage, and rely on the fallback mechanisms of the complete runtime to recover reliability (Table 3).

Table 8: Timestep-location ablation on LIBERO-10 using the Torch backend with $K = 2$ and $\delta = 0.15$. Both phase-aware and periodic fallback are disabled.

Timesteps $\mathcal{T}$	Placement	SR (%)	Acc. (%)	FR. (%)	Lat. (ms)
$\{0.10, 0.20\}$	Early	85.8	4.4	6.3	95.3
$\{0.50, 0.60\}$	Intermediate	85.0	5.7	7.5	57.2
$\{0.90, 0.95\}$	Late	58.0	67.2	85.1	29.3
$\{0.10, 0.95\}$	Cross-stage	84.6	4.8	6.7	56.3

Selecting the distance threshold. Finally, with $K = 2$ and $\mathcal{T} = \{0.90, 0.95\}$ fixed, we examine the sensitivity to the distance threshold δ. Table 7 shows the threshold trade-off on LIBERO-10, while Table 9 extends the analysis across all four LIBERO suites. Larger thresholds generally permit more speculative execution and lower latency, but can reduce task success; smaller thresholds are more conservative and push latency toward the full-path regime. The acceptance–latency trade-off is consistent across suites, whereas the success-rate response varies. LIBERO-10 exhibits the largest success-rate variation and therefore serves as the primary verifier stress test.

We use $\delta = 0.15$ as the base setting for the complete runtime, rather than as a standalone verifier-only optimum. This setting preserves useful speculative coverage and low latency, while phase-aware fallback and periodic full-path refresh recover reliability (Table 3).

Table 9: Cross-suite threshold sensitivity using the Triton backend with $K = 2$ and $\mathcal{T} = \{0.90, 0.95\}$. Phase-aware fallback and periodic refresh are disabled. The shaded row is the base setting used in the complete runtime.

δ	Success Rate (%)				Latency (ms)				Average				
	Spatial	Object	Goal	10	Spatial	Object	Goal	10	SR	Lat.	/Act	Acc.	FR.
0.30	95.8	99.2	86.6	53.8	12.0	11.1	11.8	9.9	83.9	11.1	1.0	99.7	91.3
0.25	93.6	98.4	87.6	53.8	11.9	11.3	11.8	10.0	83.4	11.3	1.0	98.8	91.4
0.20	94.8	98.6	86.4	54.3	11.8	11.3	11.9	10.3	83.5	11.2	1.0	95.1	90.9
0.15	96.0	99.2	90.8	58.4	12.3	12.6	12.5	13.3	86.1	12.7	1.4	77.5	84.9
0.10	97.2	98.8	92.6	79.4	19.5	20.7	19.5	21.1	92.0	20.2	3.0	30.5	46.6
0.05	93.8	98.8	94.4	87.2	41.0	40.2	40.9	41.6	93.5	41.0	4.1	2.0	2.5
0.00	97.0	97.6	96.2	85.6	47.4	47.2	47.2	47.9	94.1	47.5	4.1	0.0	0.0

C.2 Real-world Details

Figure 8 illustrates the real-world evaluation platform used for conveyor-belt sorting. In real-world deployment, all accepted actions, including falsely accepted drafts, are executed through a robot controller that enforces joint limits and command constraints.

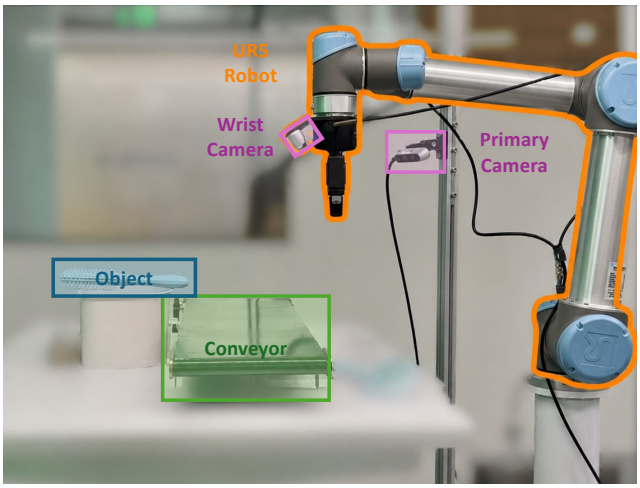


Figure 8: Real-world robot platform used for conveyor-belt sorting.

Table 10 reports the training configuration. We first fine-tune the main policy from the pre-trained π_0 checkpoint using LoRA on the collected demonstrations, and then train the draft model for the same real-world domain. For draft training, observations are taken from the real-world demonstrations, while regression targets are action chunks generated by the fine-tuned main model rather than the recorded demonstration actions.

Table 10: Training hyperparameters for real-world demonstration adaptation.

Item	**Value**
Main model fine-tuning	
Batch size	128
Training steps	20k
LR schedule	Cosine decay with 500 warmup steps
Peak / final LR	1×10^{-4} / 1×10^{-5}
Draft model training	
Optimizer	AdamW
Learning rate	2×10^{-3}
LR schedule	Cosine schedule with 1 warmup epoch and 1×10^{-5} minimum LR
Batch size	64 per GPU
Epochs	100
Gradient clipping	Global norm 1.0
Loss	Weighted Smooth L1 / Huber loss, $\beta = 1.0$
Prefix weighting	Sampled prefix up to 20 steps, $\gamma_{\text{prefix}} = 0.9$, tail weight 0.1
Checkpoint	Best validation RMS over the first 12 executable steps
Compute cost	8 NVIDIA H20 GPUs, approximately 12 hours for fine-tuning and 2 hours for draft model training

D Additional Analyses and Experiments

D.1 Verifier Validity and Visual KV Cache Analysis

Verifier validity. We evaluate whether the flow-consistency verifier reflects the discrepancy between draft actions and the full-path π_0 outputs under the same observation. We analyze 35,478 replanning rounds and 425,736 action positions on LIBERO-10. As shown in Table 11, the verifier risk is positively correlated with the draft-to-full-path discrepancy, while the accepted-prefix length is negatively correlated with the round-level discrepancy. Rejected actions also exhibit a larger mean discrepancy than accepted actions (0.372 vs. 0.211, a 76% increase). These trends hold for each of the ten tasks, and the accepted–rejected gap persists at every action index; the per-task and per-index breakdowns are omitted for space.

These results support the interpretation of verification as a local consistency criterion that ranks draft actions by their agreement with the full-path policy. They do not imply that every rejected action would cause task failure, or that every accepted action is guaranteed to preserve task-level behavior.

Table 11: Offline validation of the flow-consistency verifier on LIBERO-10. Discrepancy is measured between draft actions and full-path π_0 outputs under the same observation.

Diagnostic	Observed result
Risk vs. discrepancy	Spearman $\rho = 0.596$
Accepted-prefix length vs. round-level discrepancy	Spearman $\rho = -0.444$
Mean discrepancy: rejected actions	0.372
Mean discrepancy: accepted actions	0.211

Effect of stale visual KV cache. FLASH reuses the visual KV cache from the most recent full-path round to reduce inference overhead. To assess the impact of this reuse, we conduct a paired shadow analysis that replaces the cached KV with the KV computed from the current image, while holding the draft, robot state, noise, verification timesteps, and threshold fixed. Table 12 reports the resulting shifts in the verifier output, changes in accepted-prefix length, and agreement between prefix decisions.

On LIBERO-10, using the current-image KV changes the verifier output by an RMS of 0.315 and increases the accepted prefix by 0.187 actions on average; prefix decisions remain identical in 78.4% of replanning rounds. On LIBERO-Spatial, the corresponding output shift is 0.744, with a mean prefix increase of 0.119 actions and identical decisions in 84.1% of rounds. Thus, stale visual context does affect verification outputs, but the prefix decision remains unchanged in most rounds. Moreover, the positive mean prefix change indicates that, on average, the stale cache makes the verifier slightly more conservative rather than more permissive. These results characterize the evaluated tasks and do not provide a general bound on visual-context staleness.

Table 12: Paired shadow analysis of the stale visual KV cache.

Suite	Output shift (RMS)	Mean prefix change	Identical decisions
LIBERO-10	0.315	$+0.187$	78.4%
LIBERO-Spatial	0.744	$+0.119$	84.1%

D.2 Fallback Mechanism Analysis

Timing of full-path calls around gripper transitions. Phase-aware fallback is designed to return control to the full path near grasp and release events, which also refreshes the visual KV cache before precision-critical phases. Consistent with this design, 98.4% of stable grasp and release transitions are preceded by a full-path call within one replanning window, with mean and median lead times of 4.45 and 4 control steps, respectively.

Matched proactive-fallback budget. We compare phase-aware fallback against periodic and random fallback under the same proactive-call budget on LIBERO-10. Each method is allocated exactly 5,287 proactive calls. The methods still differ in the number of verifier-zero fallbacks, i.e., full-path calls triggered when no draft prefix is accepted, because their closed-loop trajectories diverge.

As reported in Table 13, phase-aware fallback achieves a lower success rate than periodic and random fallback, but triggers fewer verifier-zero fallbacks and fewer realized full-path calls. Relative to periodic and random fallback, it reduces verifier-zero fallbacks by 45.3% and 43.3%, respectively, and reduces realized full-path calls per 100 actions by 23.3% and 22.0%. Phase-aware fallback therefore occupies a lower-compute operating point at the matched proactive budget, rather than achieving the highest success rate. In the complete runtime, it is combined with periodic full-path refresh, which yields the best success–latency trade-off in Table 3.

Table 13: Fallback comparison on LIBERO-10 with a matched proactive budget of 5,287 calls. Verifier-zero fallbacks are additional full-path calls triggered when no draft prefix is accepted.

Method	SR (%)	Verifier-zero fallbacks	Full-path calls / 100 actions
Phase-aware	71.6	3,362	3.52
Periodic	77.8	6,142	4.59
Random	81.0	5,926	4.51

Failure-phase analysis. To examine whether phase-aware fallback addresses precision-sensitive failure modes, we label 158 of 159 verifier-only failures by their first visible failure phase. We then compare their outcomes with paired runs using phase-aware fallback, with periodic refresh disabled. Table 14 summarizes the results.

Grasp, release, and placement account for 111 of the 158 labeled failures, which is consistent with these phases being the most precision-sensitive. In the paired runs, 47 of these 111 episodes (42.3%) succeed with phase-aware fallback. However, failures in the remaining phases are recovered at a similar rate (21 of 47, 44.7%). Thus, the recovery is not specific to the phases targeted by the gripper-switch trigger; a plausible explanation is that each fallback also refreshes the visual KV cache and thereby benefits subsequent phases. Because paired trajectories can diverge after fallback decisions, these results are associations rather than controlled causal estimates.

Table 14: Paired phase-aware outcomes by first visible verifier-only failure phase on LIBERO-10.

First failure phase	Episodes	Paired successes	Success rate (%)
Approach / smooth motion	8	1	12.5
Grasp	48	17	35.4
Transport / drop	14	7	50.0
Release	11	4	36.4
Placement	52	26	50.0
Multi-stage incomplete	21	10	47.6
Other	4	3	75.0

D.3 Asynchronous Inference and Real-Time Chunking

Asynchronous inference on LIBERO. We implement a training-free asynchronous adapter on LIBERO-10 at a fixed 20 Hz control frequency. The adapter keeps one policy request in flight while executing the previous action buffer. When a request returns, it discards positions that have already elapsed, shortens the accepted prefix accordingly, merges the remaining actions while preserving the old buffer tail, and triggers full-path recovery if the buffer is exhausted.

Table 15 reports results over 500 episodes per method. Executable-action throughput counts the executable actions returned per second. Compared with asynchronous π_0, FLASH achieves $1.86\times$ the policy request throughput and $1.47\times$ the executable-action throughput. The smaller gain in executable-action throughput arises because each FLASH request returns fewer executable actions on average (7.9 vs. 10.0), due to variable accepted-prefix lengths and in-flight staleness. The median end-to-end round-trip time (RTT) is approximately halved, while the 95th-percentile RTT remains comparable, since rounds that require full-path recovery still incur full-path latency. The RTT values cluster near multiples of the 50 ms control period.

Table 15: Paired asynchronous evaluation on LIBERO-10 at 20 Hz (500 episodes per method).

Metric	π_0 async	FLASH async
Success rate (%)	86.0	87.8
Policy throughput (requests/s)	17.53	32.60
Executable-action throughput (actions/s)	175.29	257.62
End-to-end RTT, p50 (ms)	100.12	50.07
End-to-end RTT, p95 (ms)	100.28	100.06

Combining FLASH with real-time chunking. We further evaluate FLASH with real-time chunking (RTC) on the real-world toy-dog conveyor task. RTC maintains trajectory continuity during asynchronous execution, while FLASH reduces the inference latency of replanning rounds. Table 16 reports the number of successful trials out of ten at each conveyor speed.

Table 16: Toy-dog conveyor results with RTC.

Method	10 m/min	13 m/min	15 m/min
Triton-π_0 (sync)	8/10	3/10	0/10
Triton-π_0 + RTC	7/10	3/10	1/10
FLASH+Triton-π_0 (sync)	8/10	5/10	2/10
FLASH+Triton-π_0 + RTC	8/10	6/10	2/10

With RTC, FLASH+Triton-π_0 matches synchronous FLASH at 10 and 15 m/min and achieves one additional success at 13 m/min. RTC also enables Triton-π_0 to obtain one success at 15 m/min, so the observation in Table 4 that only FLASH succeeds at this speed applies to synchronous inference.

Given ten trials per condition, these results show that FLASH is compatible with RTC, but they are not sufficient to establish a significant gain from the combination. The current integration uses a fixed-buffer implementation with limited support for dynamically varying accepted-prefix lengths.

D.4 Reduced-NFE Inference and Edge Latency

Complementarity with reduced-NFE inference. Reducing the number of function evaluations (NFE) is an alternative way to shorten full-path inference. We evaluate it using the same frozen π_0 policy. Across the four LIBERO suites, NFE values of 1, 2, 4, and 10 achieve average success rates of 93.8%, 95.7%, 94.3%, and 94.1%, respectively. However, similar task-level success rates do not imply equivalent action outputs, which matters because FLASH is designed to accept only drafts that agree with the full-path policy.

To quantify this difference, we evaluate 500 held-out LIBERO observations with matched inputs and compare the executable 12×7 action prefixes against the NFE=10 reference. Table 17 shows that action discrepancy increases as NFE decreases.

Table 17: Action discrepancy relative to NFE=10 on 500 held-out LIBERO observations.

NFE	Average MSE	P95 MSE
1	0.01762	0.06554
2	0.00607	0.02112
4	0.00149	0.00248

We additionally combine FLASH with NFE=2 and NFE=4 full paths using `torch.compile` without Triton kernels. On LIBERO-10, these configurations achieve success rates of 85.8% and 86.0% at latencies of 29.6 ms and 31.1 ms, respectively, compared with 83.4% and 33.2 ms for FLASH-π_0 with NFE=10 (Tables 1 and 2). This suggests that reducing NFE and reducing the frequency of full-path inference are complementary.

Edge inference latency. Finally, we measure inference latency on NVIDIA Jetson Thor using the compiled implementation. The original π_0 requires 214.3 ms per inference, whereas FLASH requires 56.4 ms, a 3.80$\times$ speedup. This result indicates that the inference-latency benefit transfers to the evaluated edge hardware. It is not a complete end-to-end robotic deployment evaluation, which remains outside the scope of this measurement.

E Future Work

Realtime-VLA FLASH suggests several directions for future work.

First, the current verifier uses heuristic hyperparameters, including a fixed acceptance threshold and hand-picked verification timesteps. A more adaptive verifier could adjust both over the course of a trajectory, tightening verification during fine-adjustment phases, rapid observation changes, or high-curvature motions, while relaxing it during smooth free-space motion. This would generalize the intuition behind phase-aware fallback from a gripper-switch heuristic to a broader context-dependent verification rule.

Second, a tighter integration with real-time chunking is needed. RTC addresses trajectory continuity under variable inference delays, while FLASH reduces policy inference latency. Our current integration (Appendix D.3) uses a fixed buffer; a more adaptive design could jointly account for accepted-prefix lengths and policy-update timing to improve both the efficiency and the continuity of closed-loop control.

Third, FLASH may benefit edge deployment. Our measurements on NVIDIA Jetson Thor cover inference latency only, motivating evaluation in complete edge-based robotic systems. Because FLASH reduces repeated full-path calls, it could potentially also lower average memory traffic and power consumption during replanning, which would make low-latency dVLA deployment more practical on robots with limited compute, power, or thermal budgets.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims about Realtime-VLA FLASH reducing speculative-round cost and expensive full-path rounds while maintaining performance are clearly stated in the abstract and Sec 1 and are supported by both simulation and real-world experiments.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The conclusion discusses the limitation of using fixed verification thresholds. To mitigate this limitation, Appendix C.1 provides verifier-timestep and threshold ablations, including Table 7 and Table 9.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: This paper does not present formal theoretical theorems or proofs; it is primarily empirical and method-focused.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: All experimental setups, evaluation metrics, and baselines are described in detail (see Sec 4.1), including simulation and real-world conveyor-belt tasks. Appendix C.1 details verifier hyperparameter choices, and Appendix C.2 provides the real-world platform and adaptation setup.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide code and demos with the submission, together with instructions for reproducing the main simulation results. Real-world reproduction may require adapting the setup to different robot platforms.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper provides training details, action chunk sizes, replanning frequency, and evaluation protocols for both simulation and real-world experiments. Appendix A reports draft model architecture/training details, Appendix C.1 reports verifier hyperparameter selection, and Appendix C.2 reports real-world adaptation details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: Success rates are computed over repeated trials with explicitly reported sample sizes: 50 trials per simulation task and 10 trials per real-world condition. We do not report error bars or confidence intervals, so the results should be interpreted as repeated-trial empirical evidence rather than formal statistical significance tests.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper reports the main inference hardware, while Appendix A provides draft-model training and inference cost details (Table 5 and Table 6) and Appendix C.2 reports real-world training compute in Table 10.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work follows NeurIPS ethical guidelines; experiments involve robot manipulation with no human subjects or sensitive personal data.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Real-world robot deployment inherently carries physical safety risks, and faster inference may increase the need for robust safety constraints. Safe use therefore requires hardware position limits, workspace constraints, and emergency stop mechanisms.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The work does not release pre-trained models, scraped datasets, or other assets with high misuse risk.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper cites the referenced models, datasets, and prior work. The LIBERO datasets are released under CC BY 4.0, the OpenPI code and π_0 model assets are released under Apache-2.0, and the Gemma backbone is used under the Gemma Terms of Use.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The submission includes code and demo assets, and documents the model components, experimental setup, and reproduction instructions alongside the released assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: No human subjects or crowdsourcing experiments were conducted.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: Not applicable; no human participants were involved.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The research does not involve the use of LLMs as part of the core methods.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.