SAVE-WAM: SPECULATIVE ACTION VERIFICATION WITH ENVIRONMENT FEEDBACK FOR ASYNCHRONOUS WORLD-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large generative robot policies often adopt asynchronous inference to reduce execution latency, but this introduces generation discrepancy: actions are generated from observations obtained before all previously predicted actions have been executed. Although existing methods attempt to mitigate this discrepancy through calibration during action generation, there lacks feedback from newer observations affected by ongoing execution. As a result, action generation remains **"open-looped"**, allowing accumulated prediction errors and eventually causing the trajectory to deviate from the synchronous inference. Therefore, we aim to explicitly model and correct the discrepancy introduced by asynchronous action prediction in a **"closed-looped"** manner. Inspired by speculative decoding in language models, we introduce **SAVE-WAM**, a training-free post-generation feedback mechanism for asynchronous World-Action Models. SAVE-WAM treats asynchronously generated actions as speculative drafts and verifies them by measuring their deviation from the corresponding synchronous trajectory. Using updated observations during execution, the verifier detects excessively deviated trajectory, and triggers the replanning. As a plug-in mechanism, SAVE-WAM can be integrated with existing asynchronous inference methods to correct generation discrepancies with modest overhead, thereby improving the accuracy-efficiency trade-off. On LingBot-VA with FDM, SAVE-WAM improves success over unverified asynchronous execution from 2% to 42% on *Hanging Mug* and from 82.76% to 92.76% overall, while retaining 73.8% latency overlap. Moreover, verification enables the use of larger action chunks without degrading task success, providing an additional dimension for improving generation efficiency. With a larger chunk size, SAVE-WAM achieves up to $1.97\times$ higher action-generation throughput while maintaining a comparable success rate.

1 INTRODUCTION

Recent World Action Models (WAMs) (Li et al., 2026; Yuan et al., 2026; Pai et al., 2025; Ye et al., 2026c; Bi et al., 2026) have demonstrated strong performance in robotic manipulation. However, generating an action typically requires multiple denoising steps (Akbari et al., 2026; Zhao et al., 2026a), leading to substantial inference latency that prolongs task execution and hinders real-time control and practical deployment. Under the standard synchronous inference scheme, actions are generated from the current robot observation, and the observation for the next iteration is acquired only after all predicted actions have been executed. To reduce execution latency, asynchronous inference methods (Black et al., 2026; Jiang et al., 2026) overlap action generation with action execution by starting the next round of prediction before the previously generated actions have finished executing. Although this overlap improves efficiency, asynchronous inference introduces a discrepancy in the predicted action trajectory relative to its synchronous counterpart (Motubrain, 2026). Specifically, the next action chunk is generated from an outdated observation captured before all previously predicted actions have been executed, and therefore does not directly account for the effects of the remaining actions on the robot state and subsequent observations.

Existing asynchronous methods mitigate the generation–execution mismatch by improving future action prediction during generation, for example by conditioning on previously generated actions

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.

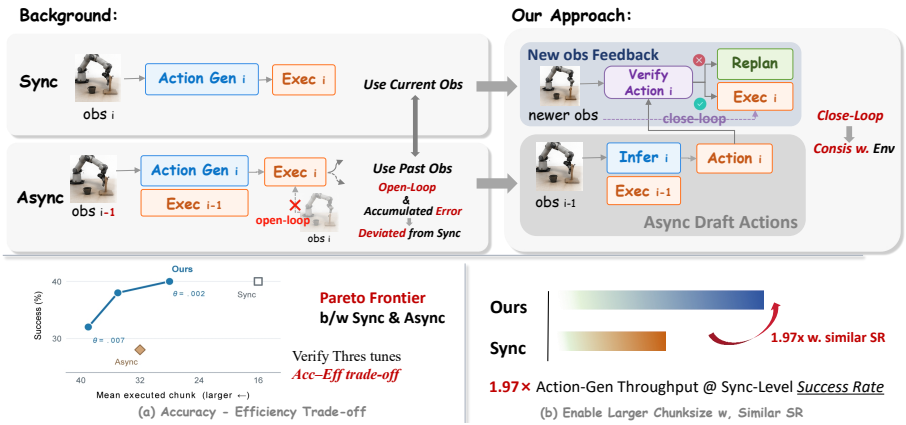


Figure 1: **Overview of SAVE-WAM.** Asynchronous inference overlaps action generation with execution to reduce latency, but actions are generated from past observations and remain open-loop with newer observation, causing trajectory deviation and accumulated error. SAVE-WAM treats asynchronously generated actions as speculative drafts and restores closed-loop feedback by verifying them with updated observations. This training-free plug-in improves the accuracy–efficiency trade-off, while enabling larger action chunks and up to $1.97\times$ higher action-generation throughput.

or explicitly predicting future observations (Black et al., 2026; Jiang et al., 2026; Li et al., 2026). However, a direct source of feedback, the updated observations available after action generation and during execution—remains under-exploited. As a result, asynchronous inference still operates in an "open-loop" manner: even small discrepancies in the predicted action trajectory can accumulate over time, eventually leading to substantial deviation from the synchronous reference trajectory.

To address this limitation, we aim to incorporate real-world feedback into asynchronous inference, thereby turning it into a "closed-loop" process. Inspired by Speculative Decoding Xia et al. (2023), we propose **SAVE-WAM**, which leverages updated observations acquired during action generation as feedback to dynamically adjust the inference process. Specifically, asynchronously generated actions are treated as drafts and verified by measuring their deviation from the corresponding synchronous reference trajectory. Rejected action tokens are discarded, triggering a replanning stage that regenerates a new action chunk conditioned on the latest observation. We carefully design each component to minimize the overhead introduced by this additional verification. In particular, the feedback (verification) module is lightweight: it performs one-step flow scoring to compare the current draft trajectory against the reference trajectory using only a single diffusion step. Furthermore, we introduce an ahead-verification scheme that overlaps verification with action generation, further hiding the verification cost and minimizing the overall inference overhead.

The SAVE-WAM draft-and-verification scheme can serve as a plug-in module for existing asynchronous inference methods, compensating for their success-rate degradation with controllable and modest overhead. In this way, it turns a single operating point into a Pareto frontier over the accuracy–efficiency trade-off. Furthermore, by using verification to prevent trajectory deviation, it can maintain task performance at larger action chunk sizes. This enables greater parallelism in action generation and, consequently, improves action-generation throughput. Our contributions as follows:

- We introduce real-world feedback into asynchronous inference through a draft-and-verify scheme, and design a lightweight feedback module together with an ahead-verification mechanism to minimize the overhead.
- SAVE-WAM is compatible with existing asynchronous inference methods, providing adjustable trade-off. Moreover, it supports larger action chunk sizes, enabling greater parallelism and higher action-generation throughput.
- While achieving success rate comparable to that of synchronous baselines, SAVE-WAM preserves most of the execution-generation overlap enabled by Lingbot FDM, also increasing the feasible action chunk size by up to $1.97\times$.

2 Related Work

World-Action Models World-Action Models (WAMs) extend robot policies by predicting actions together with future observations, conditioning on current observations. Mimic-Video (Pai et al., 2025), LingBot-VA (Li et al., 2026), DreamZero (Ye et al., 2026c), Cosmos Policy (Kim et al., 2026), and DiT4DiT (Ma et al., 2026) use imagined visual futures to inform action generation. Motus (Bi et al., 2026) jointly generates video and actions, whereas FastWAM (Yuan et al., 2026) and GigaWorld-Policy (Ye et al., 2026a) implicitly use future foresight to generate actions during the reasoning stage for efficient action prediction. These works improve how actions are generated. We study what happens afterward: whether a generated action remains valid when a new real observation arrives.

Asynchronous Robot Policy Deployment Asynchronous inference generates actions while earlier actions execute, reducing stalls caused by large policy models. RTC (Black et al., 2026) maintains continuity through committed action prefixes. VLASH (Tang et al., 2025) and FutureRTC (Jiang et al., 2026) train the model to enable it to generate actions in advance. WAM-based approaches often exploit explicit predictions of future observations to facilitate asynchronous inference, as in FDM (Li et al., 2026) and AHA-WAM (Cai et al., 2026). These methods primarily improve the model's ability to generate actions asynchronously at generation time, whereas our approach is complementary: it can be combined with them to validate asynchronously pre-planned actions at execution time using feedback from updated observations.

Adaptive Action Execution Adaptive execution chooses when to stop following a planned chunk and replan. Prior methods mainly based on synchronous inference, using action uncertainty (Liang et al., 2026; Feng et al., 2026), internal model features (Wang et al., 2026a; Xu et al., 2026), imagined–real observation consistency (Wang et al., 2026b; Pan et al., 2026b), reinforcement learning signals (Zhao et al., 2026b), or task progress (Ye et al., 2026b). We likewise use feedback to determine the executed horizon, but target asynchronous open-loop problem. A training-free verifier checks successive segments of a previously generated draft against updated observations, allowing valid actions to stream through and rejecting the remaining suffix when they become inconsistent.

Speculative Inference and Verification Speculative decoding (Leviathan et al., 2023; Xia et al., 2023) accelerates generation by proposing drafts, accepting verified prefixes, and regenerating rejected suffixes. Related acceleration ideas extend to continuous generative models (De Bortoli et al., 2025; Pan et al., 2026a). We apply the draft-and-verify principle to robot execution, where validity depends on the changing physical environment feedback.

3 Preliminaries

Feedback in robot execution. In feedback control (Doyle et al., 2013), control inputs change the state of a plant, and the resulting measurements are used to update subsequent inputs. For a robot policy, the robot and its environment form the plant:

$$x_{t+1} = f(x_t, a_t), \qquad o_{t+1} = q(x_{t+1}), \tag{1}$$

where a_t is the executed action, x_t is the environment state, and o_t is the observation provided to the policy as feedback. The policy maps an observation to a chunk of H future actions,

$$\mathbf{A}_t = \pi(o_t) = [a_t, \ldots, a_{t+H-1}]. \tag{2}$$

In synchronous inference, the policy waits for the previous chunk to finish, observes the resulting state, and then generates the next chunk. This allows action generation to use the latest observation, but introduces an inference pause. Asynchronous inference instead generates actions before all previously predicted actions have been executed. It therefore lacks feedback from newer observations affected by ongoing execution, leaving the pre-generated actions effectively open-looped and allowing prediction errors to accumulate.

We describe such feedback through its *input*, *module*, and *output*:

$$z = \mathcal{F}(o^{\text{new}}), \qquad \mathbf{A}^{\text{exec}} = \mathcal{U}(\mathbf{A}^{\text{pending}}, z). \tag{3}$$

Here, o^{new} is the feedback input, $\mathcal{F}$ is the feedback module that produces a signal z, and $\mathcal{U}$ determines the feedback output by controlling which pending actions enter the execution queue.

Latency in chunked execution. Let Δt be the controller period and $T_H = H\Delta t$ the execution duration of an H-action chunk. Let T_{infer} denote the latency of policy inference, and T_{halt} the uncovered waiting time between consecutive chunks. A synchronous policy places the inference latency on the execution critical path,

$$T_{\text{halt}}^{\text{sync}} = T_{\text{infer}}, \tag{4}$$

whereas asynchronous execution overlaps inference with ongoing execution,

$$T_{\text{halt}}^{\text{async}} = \max(0, T_{\text{infer}} - T_H). \tag{5}$$

A larger chunk improves efficiency in two ways. First, it provides a longer execution window to hide policy inference. Second, each policy call generates more executable actions, reducing the inference cost per action to approximately T_{infer}/H. When T_{infer} grows slowly with H, increasing the chunk size can therefore reduce the effective inference cost per action and increase action-generation throughput. However, larger chunks also rely on older observations for longer, increasing the discrepancy between generation and execution. This efficiency–accuracy trade-off motivates adding feedback that enables larger chunks while correcting stale actions before execution.

4 METHOD

Overview. We introduce a feedback-gated framework that introduces closed-loop feedback into asynchronous action execution. Compared with synchronous chunking, asynchronous inference generates actions further ahead of their execution, thereby extending the effective open-loop interval. Our method closes this loop through three components. Section 4.1 defines the *feedback input & output*: how newer observations provide feedback to ongoing action execution. Section 4.2 defines the *feedback module*: a low-cost, one-step flow score that determines whether the generated actions remain valid. Section 4.3 further reduces the overhead introduced by the feedback mechanism by delivering recent real observations to the feedback module without stalling action execution. The system can therefore accept more actions in predictable states while triggering earlier replanning when new feedback reveals a mismatch.

4.1 FEEDBACK-GATED ACTION EXECUTION

Asynchronous inference lacks feedback from newer observations during execution, leaving pre-generated actions effectively open-looped. We therefore introduce *feedback-gated action execution* that uses newer observations to verify draft actions, defining the feedback *input* as the newer observation and the feedback *output* as whether the pending actions should be executed.

Given a generation-time observation o^{gen}, the asynchronous policy first produces a speculative draft:

$$\mathbf{A}^{\text{d}} = \pi(o^{\text{gen}}). \tag{6}$$

As shown in Figure 2, we decouple action proposal from action commitment. The policy determines *what* actions to propose, while feedback determines *whether* draft actions remain valid under newer observations. Given the latest observation at execution-time o^{exec}, the verifier produces

$$g = \mathcal{G}(A^d, o^{\text{exec}}), \qquad A^{\text{exec}} = \begin{cases} A^d, & g = 1, \\ \varnothing, & g = 0. \end{cases} \tag{7}$$

Accepted actions are executed, while rejected actions are discarded and replanned from a recent observation. Repeating this process turns a fixed draft horizon into a dynamic accepted prefix, allowing longer reuse in predictable states and earlier replanning when a mismatch occurs.

The remaining challenge is to provide this feedback with minimal overhead. Sections 4.2 and 4.3 reduce the verification cost and avoid stalling action execution, respectively.

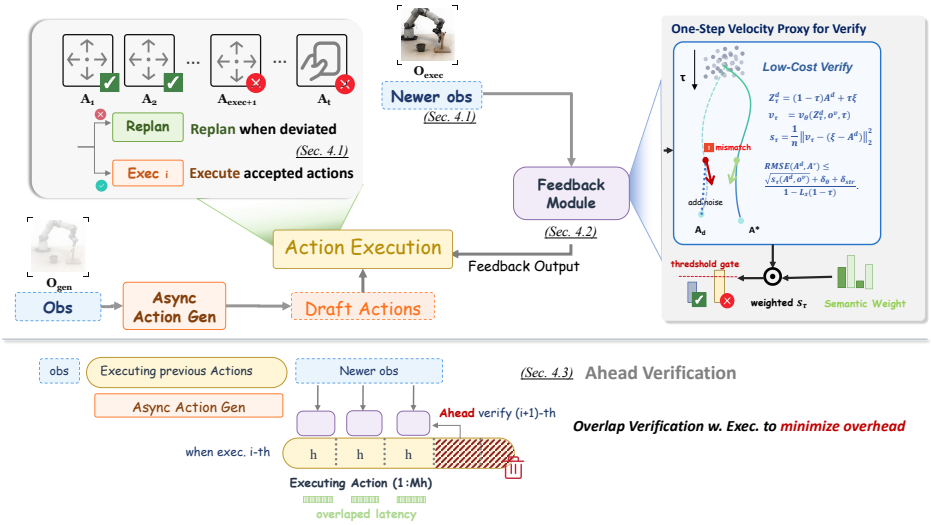


Figure 2: **Feedback mechanism of SAVE-WAM.** The policy generates speculative action chunks from a planning observation, while the feedback module decides whether pending actions remain executable under newer execution-time observations. Valid actions are committed to the queue; mismatched actions are discarded and followed by replanning, yielding a dynamic accepted prefix.

4.2 Low-Cost Draft Verification

The feedback mechanism above requires determining whether a draft action segment should still be executed under a newer observation. Directly regenerating the corresponding actions provides an oracle verification signal, but largely removes the efficiency benefit of asynchronous execution. We therefore introduce a low-cost draft verification method.

Oracle verification. Let $A^{\mathrm{d}} \in \mathbb{R}^{h \times d_a}$ be an h-step draft segment awaiting verification, and let o^{v} be a newer real observation. An oracle reruns the policy on o^{v}, as in synchronous planning, and directly compares the draft segment with newly generated reference actions:

$$A^{\star} = \pi_h(o^{\mathrm{v}}; \xi), \qquad d^{\mathrm{oracle}} = \frac{\|A^{\mathrm{d}} - A^{\star}\|_2}{\sqrt{n}}, \tag{8}$$

where $n = hd_a$, π_h extracts the corresponding h-step block from the full policy output, and ξ is coupled flow noise. A small discrepancy indicates that the draft remains close to the sync policy decision under the updated observation. However, computing $A^{\star}$ requires full multi-step inference at a cost comparable to replanning.

One-step verification. We replace oracle regeneration with a single vector-field evaluation. We first noise the draft segment to flow time τ:

$$Z_\tau^{\mathrm{d}} = (1-\tau)A^{\mathrm{d}} + \tau\xi, \qquad v_\tau^{\mathrm{d}} = \frac{\partial Z_\tau^{\mathrm{d}}}{\partial \tau} = \xi - A^{\mathrm{d}}. \tag{9}$$

At the same state, we replace the original planning observation with o^{v} and inference the velocity of action denoising at flow time τ:

$$\hat{v}_\tau = v_\theta(Z_\tau^{\mathrm{d}}, o^{\mathrm{v}}, \tau), \qquad s_\tau(A^{\mathrm{d}}, o^{\mathrm{v}}) = \frac{1}{n}\left\|\hat{v}_\tau - v_\tau^{\mathrm{d}}\right\|_2^2. \tag{10}$$

If the draft remains compatible with the updated observation, the predicted velocity should stay close to the velocity associated with the noised draft. This provides a low-cost verification signal without generating a new action trajectory. In practice, we account for heterogeneous action semantics and scales through dimension-wise weighting: discrete gripper open/close commands are treated as hard consistency constraints, while continuous dimensions are normalized by their average magnitudes.

Theorem 1 (Action-error bound for one-step verification)**.** *Let $v^\star$ denote the target vector field conditioned on o^{v}. Assume that the model approximation error is bounded by δ_θ, $v^\star$ is L_x-Lipschitz with respect to the action state, and its deviation from the linear flow between $A^\star$ and ξ is bounded by δ_{str}. For any τ satisfying $L_x(1-\tau) < 1$,*

$$d^{\mathrm{oracle}} \leq \frac{\sqrt{s_\tau(A^{\mathrm{d}}, o^{\mathrm{v}})} + \delta_\theta + \delta_{\mathrm{str}}}{1 - L_x(1-\tau)}. \tag{11}$$

Theorem 1 shows that the one-step score controls the same action discrepancy between the draft and synchronous reference measured by the oracle. Its proof is given in Appendix A.1, and their empirical correlation is further evaluated in Appendix A.2. We therefore instantiate the feedback decision in Equation equation 7 as

$$g(A^{\mathrm{d}}, o^{\mathrm{v}}) = \mathbb{I}\left[s_\tau(A^{\mathrm{d}}, o^{\mathrm{v}}) < \varepsilon_{\mathrm{ver}}\right], \tag{12}$$

which accepts actions consistent with the new observation and discards inappropriate actions.

4.3 Speculative Ahead Verification

Although the verifier in Section 4.2 greatly reduces verification cost, performing verification immediately before execution places its latency T_{ver} on execution critical path. However, since verification is now much cheaper than full action generation, it only needs to be moved a few actions ahead to hide this latency. We therefore verify upcoming actions while the current ones are still being executed, using a recent observation and finishing before the verified actions are needed to execute.

Large proposal blocks, small verification segments. We preserve the long generation horizon of asynchronous inference, but partition each draft into M consecutive segments:

$$\mathbf{A}^{\mathrm{d}} = [A_1^{\mathrm{d}}, \ldots, A_M^{\mathrm{d}}], \qquad H = Mh, \tag{13}$$

where each segment A_j^{d} contains h actions. The large generation horizon H reduces how often full policy inference is needed, while the smaller segment size h enables more frequent feedback within the generated draft.

When executing A_j^{d}, the robot acquires observation o_j^{v} and verifies the next segment in parallel:

$$g_{j+1} = g(A_{j+1}^{\mathrm{d}}, o_j^{\mathrm{v}}). \tag{14}$$

Compared with the observation used to generate $\mathbf{A}^{\mathrm{d}}$, o_j^{v} is much closer to the execution time of A_{j+1}^{d}. Since the current segment provides an execution window $T_h = h\Delta t$, verification introduces no additional stall when

$$T_{\mathrm{ver}} \leq T_h. \tag{15}$$

Pipelined feedback and replanning. If $g_{j+1} = 1$, the next segment is accepted and executed immediately after the current one, while verification moves on to the following segment. If $g_{j+1} = 0$, the remaining unexecuted actions $[A_{j+1}^{\mathrm{d}}, \ldots, A_M^{\mathrm{d}}]$ are discarded and the policy replans from a recent observation. In this way, low-cost verification runs in parallel with action execution, minimizing the additional overhead introduced by feedback. This largely maintains the latency benefit of asynchronous inference while allowing newer observations to provide feedback.

5 Experiments

5.1 Experimental Setup

Backbones. We evaluate our method on two flow-based World Action Models with different asynchronous mechanisms. LingBot-VA uses its native FDM to predict future observations during asynchronous inference (Li et al., 2026), with 5 video and 10 action denoising steps. Fast-WAM is paired with RTC, which conditions asynchronous generation on committed actions, with its default denoising steps (Yuan et al., 2026; Black et al., 2026).

Table 1: Results on RoboTwin for two policy backbones. E and H denote the clean (easy) and random (hard) settings. SR is success rate (%), Chunk is the executed chunk size, and Time is the average completion time (s) over successful episodes. T_{inf} is the average total model-inference time (s) over successful episodes. Full results are in Appendix A.3.

LingBot-VA

Task	Baseline	SR (%) ↑		Chunk ↑		Time (s) ↓	T_{inf} (s) ↓
		E	H	E	H		
Hanging Mug	Ours (Async-64+Verify)	**38**	42	35.03	28.03	139.00	47.09
	FDM Async-32	26	30	32.00	32.00	128.69	50.52
	FDM Async-64	4	2	64.00	64.00	121.28	25.77
	Sync-16	30	**46**	16.00	16.00	156.61	89.61
	AAC	20	16	30.65	30.83	135.28	53.82
Place Can Basket	Ours (Async-64+Verify)	82	**88**	34.70	29.02	123.53	37.49
	FDM Async-32	74	80	32.00	32.00	113.82	39.84
	FDM Async-64	76	80	64.00	64.00	105.45	20.33
	Sync-16	**84**	86	16.00	16.00	133.70	70.67
	AAC	70	80	30.19	30.34	123.94	43.09
	...						
Avg. (50 tasks)	Ours (Async-64+Verify)	92.76	89.20	31.55	28.41	91.43	30.99
	FDM Async-32	85.64	83.68	32.00	32.00	80.31	30.99
	FDM Async-64	82.76	83.20	64.00	64.00	76.66	15.81
	Sync-16	**92.80**	**90.68**	16.00	16.00	97.97	54.97
	AAC	85.32	82.08	27.61	27.52	86.76	35.48

FastWAM

Task	Baseline	SR (%) ↑		Chunk ↑		Time (s) ↓	T_{inf} (s) ↓
		E	H	E	H		
Move Stapler Pad	Ours (RTC + Verify)	**90**	**78**	18.73	18.40	34.02	8.69
	RTC	78	76	19.25	19.24	28.09	8.31
	Sync-12	76	74	12.00	12.00	42.81	11.42
	AAC	86	68	16.12	15.94	34.36	10.03
Put Object Cabinet	Ours (RTC + Verify)	**94**	**92**	19.48	18.61	56.20	14.29
	RTC	90	80	19.50	19.64	52.27	13.54
	Sync-12	88	86	12.00	12.00	63.85	18.99
	AAC	90	90	16.55	15.95	60.86	16.68
	...						
Avg. (50 tasks)	Ours (RTC + Verify)	**92.52**	**92.72**	18.97	18.28	41.02	10.94
	RTC	90.92	90.96	18.98	18.94	34.97	10.52
	Sync-12	92.24	91.88	12.00	12.00	49.79	14.43
	AAC	91.72	91.76	16.50	15.89	43.72	12.67

Benchmarks. We evaluate all tasks in RoboTwin 2.0 (Chen et al., 2025) under both clean and randomized settings, with 50 rollouts per task. On LIBERO (Liu et al., 2023), we evaluate all tasks for FastWAM, using 50 rollouts per task. For LingBot-VA, we report results only on LIBERO-10, as the publicly released checkpoint is trained exclusively on this suite.

Baselines. We compare against synchronous execution, each backbone's native asynchronous scheme, and AAC adapted to asynchronous execution (Liang et al., 2026). Labels such as Async-64 denote asynchronous execution with a action chunk size of 64. Our method uses the same asynchronous backbones, adding pipelined verification and feedback-triggered replanning.

Metrics. We report **Success Rate**, **Average Executed Chunk Size**, **Inference Latency** (T_{infer}), and **Task Completion Time**. Average Executed Chunk Size measures the number of actions executed per full policy inference and reflects action-generation throughput. T_{infer} measures the latency of policy inference. Task Completion Time measures the end-to-end wall-clock time, including inference and execution, and is averaged over successful rollouts.

5.2 Experiments Results

Recovering asynchronous success with verification overhead. Tables 1 and 2 show that verification recovers much of the success lost in asynchronous execution, at the cost of extra verification and replanning. On RoboTwin, SAVE-WAM improves LingBot-VA over FDM Async-64 by 10.00 and 6.00 percentage points in the clean and random settings, while increasing completion time from 76.66 to 91.43s. For FastWAM, success improves by 1.60 and 1.76 points over RTC, with time increasing from 34.97 to 41.02s. On LIBERO-10, LingBot-VA improves from 69.40% to 95.80%,

Table 2: LIBERO results with model-specific baselines. SR is success rate, Chunk is the executed chunk size, and Time is the average successful-episode completion time. For LingBot-VA, we report results only on LIBERO-10, as the publicly released checkpoint is trained exclusively on this suite.

FastWAM

Suite	Ours			Sync-8			RTC			AAC		
	SR (%) ↑	Chunk ↑	Time (s) ↓	SR (%) ↑	Chunk ↑	Time (s) ↓	SR (%) ↑	Chunk ↑	Time (s) ↓	SR (%) ↑	Chunk ↑	Time (s) ↓
libero_10	90.00	14.84	10.440	**92.00**	8.00	12.950	75.80	15.72	9.950	76.20	16.58	13.440
libero_goal	93.00	15.58	5.820	**93.80**	8.00	6.010	91.00	15.14	5.030	88.20	16.58	6.140
libero_spatial	94.60	15.22	5.960	**95.40**	8.00	6.610	89.80	15.23	5.310	86.00	16.61	6.190
libero_object	99.00	16.06	6.380	**99.60**	8.00	7.060	93.00	15.34	5.690	89.00	16.56	7.240
Overall	94.15	15.43	7.150	95.20	8.00	8.160	87.40	15.35	6.495	84.85	16.58	8.253

LingBot-VA

Suite	Ours (Async-16+Verify)			FDM Async-8			FDM Async-16			Sync-8			AAC		
	SR (%) ↑	Chunk ↑	Time (s) ↓	SR (%) ↑	Chunk ↑	Time (s) ↓	SR (%) ↑	Chunk ↑	Time (s) ↓	SR (%) ↑	Chunk ↑	Time (s) ↓	SR (%) ↑	Chunk ↑	Time (s) ↓
libero_10	95.80	11.45	57.159	86.60	8.00	57.293	69.40	16.00	54.873	**96.00**	8.00	63.439	84.00	7.58	61.764

while time increases from 54.87 to 57.16s. Overall, post-generation feedback moves asynchronous execution toward higher-success operating points with moderate extra time cost.

Tuning the success–efficiency trade-off. The verification threshold controls how much of a speculative draft is accepted before replanning, providing an adjustable trade-off between success and action reuse. As shown in Figure 3, a threshold of 0.002 reaches 40% success with an average executed chunk of 28. Increasing it to 0.004 gives 38% success with a chunk size of 35, while 0.007 gives 32% success with a chunk size of 39. Thus, stricter verification replans earlier and favors success, while looser verification executes more actions per full policy inference. SAVE-WAM therefore provides a range of operating points rather than a fixed execution horizon.

Higher efficiency at near-synchronous success rates. Compared with synchronous execution, SAVE-WAM supports larger effective action chunks while keeping similar success rates. On RoboTwin-clean, LingBot-VA achieves 92.76% versus 92.80% for Sync-16, while increasing the average executed chunk from 16 to 31.55 actions (1.97×). FastWAM achieves 92.52% versus 92.24% for Sync-12, with 18.97 versus 12 actions per chunk (1.58×). Completion time also decreases from 97.97 to 91.43s and from 49.79 to 41.02s, respectively. The same trend appears on LIBERO: FastWAM reaches 94.15% versus 95.20% for Sync-8, with an average chunk of 15.43 versus 8 and completion time of 7.15 versus 8.16s. These results show that verification enables more executed actions per full policy inference while keeping success close to synchronous execution.

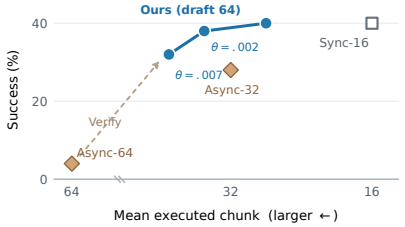


Figure 3: **Chunk-size–success trade-off on *Hanging Mug*.** Varying the verification threshold θ recovers success from Async-64 and traces the high-success Pareto frontier.

Comparison with adaptive action chunking. Table 1 compares SAVE-WAM with AAC under asynchronous execution. On RoboTwin-clean, LingBot-VA achieves 92.76% success with SAVE-WAM versus 85.32% with AAC, while increasing the average chunk from 27.61 to 31.55 actions, at a higher completion time of 91.43 versus 86.76s. For FastWAM, SAVE-WAM improves both success and efficiency: success rises from 91.72% to 92.52%, the average chunk from 16.50 to 18.97, and completion time drops from 43.72 to 41.02s. These results show that observation-based verification provides a stronger success–efficiency trade-off than the evaluated adaptive-chunking baseline, although the time cost varies by backbone.

Table 3: Ablation study on two challenging RoboTwin-clean tasks using LingBot-VA with an Async-64 draft. *Obs.* denotes the observation used for feedback: generation-time (*Gen.-time*), execution-time (*Exec.-time*), or an observation obtained x actions ahead of execution (*Ahead-x*). Latency and chunk size are averaged over successful episodes.

Variant	Input	Module	Output		Pipeline	*Hanging Mug*			*Place Can Basket*		
	Obs.	Verifier	Prefix	Replan	Overlap	SR↑	Lat.↓	Chunk↑	SR↑	Lat.↓	Chunk↑
Async Baseline (64)	Gen.-time	–	Fixed	×	–	4.00	121.281	64.00	76.00	105.451	64.00
+ Feedback (Sec. 4.1)	Exec.-time	Oracle	Dynamic	✓	×	**40.00**	184.074	**38.73**	**86.00**	159.475	**37.05**
+ Low-cost Verify (Sec. 4.2)	Exec.-time	One-step	Dynamic	✓	×	38.00	161.565	35.61	84.00	130.840	34.65
+ Ahead Verify (Sec. 4.3)	Ahead-x	One-step	Dynamic	✓	✓	38.00	**138.995**	35.03	82.00	**123.526**	34.70

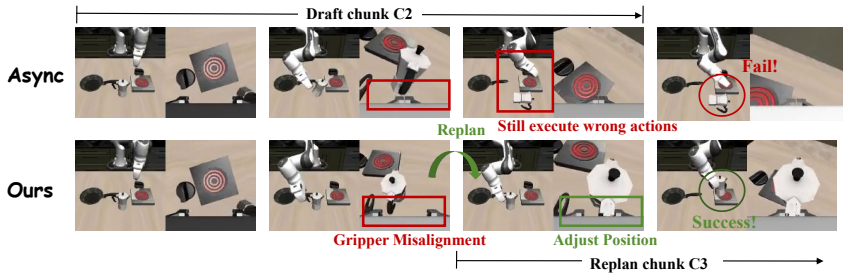


Figure 4: **Asynchronous versus feedback-gated execution.** After gripper misalignment, the asynchronous baseline continues with stale actions and fails; SAVE-WAM detects the mismatch, replans, and completes the task.

5.3 Ablation Study

We ablate SAVE-WAM on two challenging RoboTwin-clean tasks, *Hanging Mug* and *Place Can Basket*, using LingBot-VA with an Async-64 draft. Table 3 studies the three elements of feedback. The *input* determines when the feedback observation is obtained: the asynchronous baseline uses the original generation-time observation, synchronous verification uses the latest execution-time observation, and ahead verification uses an observation acquired $x = 8$ actions before execution. The *module* compares oracle regeneration with our one-step flow-based verifier, while the *output* forms a dynamic accepted prefix by retaining valid actions and replanning after rejection.

Feedback improves success over Async-64 from 4% to 40% on *Hanging Mug* and from 76% to 86% on *Place Can Basket*. Replacing the oracle with our low-cost verifier causes only a 2-point drop on both tasks, indicating a small approximation gap. Ahead verification further maintains similar success ($38\%/82\%$) while reducing successful-episode latency from $161.565/130.840$ s to $138.995/123.526$ s by overlapping most verification computation with execution of the current action segment.

5.4 Qualitative Example

Figure 4 shows the consequence of executing pre-generated actions without updated feedback. The asynchronous baseline attempts a misaligned grasp and continues as if it succeeded, ultimately failing. SAVE-WAM instead detects that the draft is deviated with the new observation, rejects its invalid suffix, and replans to correct the gripper position before completing the task.

6 Conclusion

We introduced SAVE-WAM, a training-free, post-generation feedback framework that verifies asynchronous WAM actions using newer observations and adapts the accepted chunk length. Low-cost ahead verification limits overhead. On RoboTwin and LIBERO, it achieves longer executed chunks with success rates close to synchronous inference, improving the accuracy–efficiency trade-off.

AI USE STATEMENT

Generative AI tools were used to assist with language editing and improving the clarity and readability of the manuscript. We also used generative AI to help search for, identify, and organize relevant literature to expand our research horizon and let us be able to compare our approach with prior work. All cited references and the corresponding technical claims were manually checked by the authors against the original sources.

Generative AI was not used to originate the research idea, design the proposed method or experiments, conduct the experiments, or determine the interpretation and conclusions of the results. All research decisions and scientific claims were made and verified by the authors. We reviewed all AI-assisted text and literature suggestions and take full responsibility for the final content of this work, including all text, claims, citations, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we will provide the implementation of our method and evaluation pipeline in the supplementary materials, together with a README containing instructions, configurations, and commands for reproducing our main results.

REFERENCES

Arman Akbari, Ci Zhang, Arash Akbari, Lin Zhao, Yixiao Chen, Weiwei Chen, Xuan Zhang, Geng Yuan, and Yanzhi Wang. Flash-wam: Modality-aware distillation for world action models. *arXiv preprint arXiv:2606.05254*, 2026.

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, et al. Motus: A unified latent action world model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 35101–35113, 2026.

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38:33383–33407, 2026.

Jisong Cai, Long Ling, Shiwei Chu, Zhongshan Liu, Jiayue Kang, Zhixuan Liang, Wenjie Xu, Yinan Mao, Weinan Zhang, Xiaokang Yang, et al. Aha-wam: Asynchronous horizon-adaptive world-action modeling with observation-guided context routing. *arXiv preprint arXiv:2606.09811*, 2026.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025.

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

John C Doyle, Bruce A Francis, and Allen R Tannenbaum. *Feedback control theory*. Courier Corporation, 2013.

Xiangdong Feng, Yuxuan Cheng, Chen Shi, Boyao Han, Yuxuan Yan, Yitong Hong, Zhuotao Tian, and Li Jiang. Denoising tells when to replan: Denoising-variance adaptive chunking for flow-based robot policies. *arXiv preprint arXiv:2606.03847*, 2026.

Hai Jiang, Yixian Zou, Binbin Liang, Boqian Liu, Fanman Meng, and Shuaicheng Liu. Futurertc: Real-time robot execution with anticipatory-conditioned action chunking. *arXiv preprint arXiv:2607.24008*, 2026.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, et al. Cosmos policy: Fine-tuning video models for visuomotor control and planning. *arXiv preprint arXiv:2601.16163*, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Fei Han, Mingrui Yu, Zelin Gao, Nan Xue, Xing Zhu, et al. Causal world modeling for robot control. *arXiv preprint arXiv:2601.21998*, 2026.

Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. *arXiv preprint arXiv:2604.04161*, 2026.

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.

Teli Ma, Jia Zheng, Zifan Wang, Chunli Jiang, Andy Cui, Junwei Liang, and Shuo Yang. Dit4dit: Jointly modeling video dynamics and actions for generalizable robot control. *arXiv preprint arXiv:2603.10448*, 2026.

Motubrain. World action models in real time: An empirical study of smooth execution via asynchronous deployment. *arXiv preprint arXiv:2608.01880*, 2026.

Jonas Pai, Liam Achenbach, Victoriano Montesinos, Benedek Forrai, Oier Mees, and Elvis Nava. mimic-video: Video-action models for generalizable robot control beyond vlas. *arXiv preprint arXiv:2512.15692*, 2025.

Jiayi Pan, Jiaming Xu, Yongkang Zhou, and Guohao Dai. Specdiff: Accelerating diffusion model inference with self-speculation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 8233–8241, 2026a.

Yi Pan, Miao Pan, Qi Lu, Jiaming Huang, Man Zhang, Siteng Huang, Xin Li, Jie Zhang, Yongliang Shen, Xuhong Zhang, et al. Vla-corrector: Lightweight detect-and-correct inference for adaptive action horizon. *arXiv preprint arXiv:2607.01804*, 2026b.

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. *arXiv preprint arXiv:2512.01031*, 2025.

Haoxuan Wang, Gengyu Zhang, Yan Yan, Ramana Rao Kompella, and Gaowen Liu. Vla knows its limits: Adaptive execution horizons for robot policies. *arXiv preprint arXiv:2602.21445*, 2026a.

Rui Wang, Yue Zhang, Jiehong Lin, Kuncheng Luo, Jianan Wang, Zhongrui Wang, and Xiaojuan Qi. When to trust imagination: Adaptive action execution for world action models. *arXiv preprint arXiv:2605.06222*, 2026b.

Heming Xia, Tao Ge, Peiyi Wang, Si-Qing Chen, Furu Wei, and Zhifang Sui. Speculative decoding: Exploiting speculative execution for accelerating seq2seq generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3909–3925, 2023.

Runze Xu, Xiaolong Shan, Shuang Dai, Yu Wang, and Jincheng Yu. Knowing when to stop: Adaptive action chunking via internal cross-attention dynamics in vlas. *arXiv preprint arXiv:2609.00908*, 2026.

Angen Ye, Boyuan Wang, Chaojun Ni, Guan Huang, Guosheng Zhao, Hao Li, Hengtao Li, Jie Li, Jindi Lv, Jingyu Liu, et al. Gigaworld-policy: An efficient action-centered world–action model. *arXiv preprint arXiv:2603.17240*, 2026a.

Feng Ye, Yiming Zhao, Yong Yu, Hongxu Zhou, Yong Pan, Yuan Xue, Peng Jia, and Chuanmin Jia. Rethink before you execute: Adaptive execution for world action models. *arXiv preprint arXiv:2608.09492*, 2026b.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026c.

Tianyuan Yuan, Zibin Dong, Yicheng Liu, and Hang Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026.

Weiheng Zhao, Haoyi Jiang, Xin Shi, Liu Liu, Fan Huang, Zhizhong Su, Wei Sui, and Xinggang Wang. Faster-wam: Efficient inference-time future conditioning for robust world action models. *arXiv preprint arXiv:2608.04404*, 2026a.

Yuchi Zhao, Miroslav Bogdanovic, Arjun Sohal, Liyu Tao, Kourosh Darvish, Alán Aspuru-Guzik, Florian Shkurti, and Animesh Garg. Dynamic execution horizon prediction for chunk-based robot policies. *arXiv preprint arXiv:2606.11408*, 2026b.

A Appendix

A.1 Proof of the One-Step Validity Bound

Proof. For brevity, define

$$e_A = \mathrm{RMSE}(\hat{A}, A^\star) = \frac{1}{\sqrt{n}} \left\| \hat{A} - A^\star \right\|_2 . \tag{16}$$

Because the draft and oracle trajectories use the same noise ξ,

$$\hat{A} - A^\star = (\xi - A^\star) - (\xi - \hat{A}). \tag{17}$$

Using $v_\tau^{\mathrm{d}} = \xi - \hat{A}$ and inserting the predicted and target vector fields gives

$$\begin{aligned} e_A &= \frac{1}{\sqrt{n}} \left\| (\xi - A^\star) - v_\tau^{\mathrm{d}} \right\|_2 && (18) \\ &\le \frac{1}{\sqrt{n}} \left\| v_\tau^{\mathrm{d}} - \hat{v}_\tau \right\|_2 \\ &\quad + \frac{1}{\sqrt{n}} \left\| \hat{v}_\tau - v^\star(Z_\tau^{\mathrm{d}}, o_{\mathrm{exec}}, \tau) \right\|_2 \\ &\quad + \frac{1}{\sqrt{n}} \left\| v^\star(Z_\tau^{\mathrm{d}}, o_{\mathrm{exec}}, \tau) - v^\star(Z_\tau^\star, o_{\mathrm{exec}}, \tau) \right\|_2 \\ &\quad + \frac{1}{\sqrt{n}} \left\| v^\star(Z_\tau^\star, o_{\mathrm{exec}}, \tau) - (\xi - A^\star) \right\|_2 . && (19) \end{aligned}$$

The first term is exactly the square root of the verification score:

$$\frac{1}{\sqrt{n}} \left\| v_\tau^{\mathrm{d}} - \hat{v}_\tau \right\|_2 = \sqrt{s_\tau(\hat{A}, o_{\mathrm{exec}})}. \tag{20}$$

The second and fourth terms are bounded by δ_θ and δ_{str}, respectively. Moreover,

$$Z_\tau^{\mathrm{d}} - Z_\tau^\star = (1 - \tau)(\hat{A} - A^\star). \tag{21}$$

The Lipschitz assumption therefore gives

$$\begin{aligned} &\frac{1}{\sqrt{n}} \left\| v^\star(Z_\tau^{\mathrm{d}}, o_{\mathrm{exec}}, \tau) - v^\star(Z_\tau^\star, o_{\mathrm{exec}}, \tau) \right\|_2 && (22) \\ &\qquad \le L_x(1 - \tau) e_A. && (23) \end{aligned}$$

Substituting these bounds into Equation equation 19 yields

$$e_A \le \sqrt{s_\tau(\hat{A}, o_{\mathrm{exec}})} + \delta_\theta + \delta_{\mathrm{str}} + L_x(1 - \tau) e_A. \tag{24}$$

When $L_x(1 - \tau) < 1$, rearranging gives

$$e_A \le \frac{\sqrt{s_\tau(\hat{A}, o_{\mathrm{exec}})} + \delta_\theta + \delta_{\mathrm{str}}}{1 - L_x(1 - \tau)}, \tag{25}$$

□

A.2 One-Step Velocity Consistency as a Verification Signal

Our low-cost verifier is motivated by the action flow-matching objective. Given a draft action a_d, noise ϵ, and a fixed noise level σ, let $x_\sigma = (1 - \sigma)a_d + \sigma\epsilon$. The target velocity associated with the draft is $v^\star = \epsilon - a_d$. A single model evaluation produces $v_\theta(x_\sigma; c)$ under verification context c, and we define the one-step consistency error as

$$s_{\mathrm{1step}} = \mathrm{RMSE}\,\mathcal{A}\big(v\theta(x_\sigma; c), , \epsilon - a_d\big), \tag{26}$$

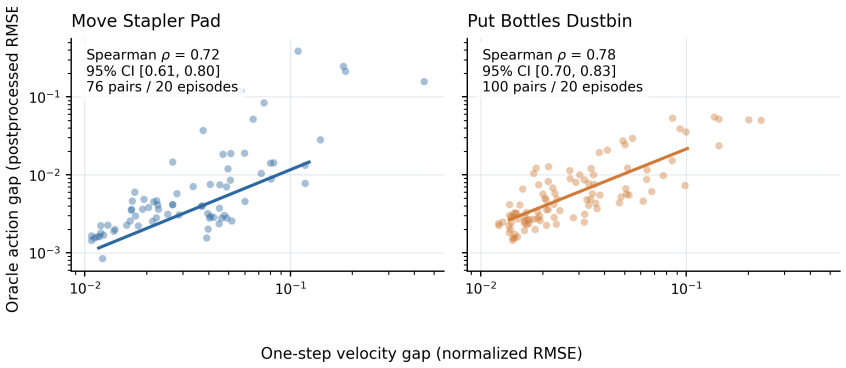


Figure 5: **One-step velocity error vs. full-oracle action discrepancy on RoboTwin-Clean.** Each point is one paired verification decision; lines show robust log–log trends.

computed over the active action dimensions. The action implied by this one-step velocity estimate satisfies

$$\tilde{a}_0 - a_d = \big(x_\sigma - \sigma v_\theta(x_\sigma; c)\big) - a_d = -\sigma\big(v_\theta(x_\sigma; c) - v^\star\big). \tag{27}$$

Therefore, velocity inconsistency is directly proportional to the magnitude of the one-step correction to the draft. This motivates s_{1step} as a low-cost proxy for whether a draft would disagree with an expensive, fully denoised oracle action.

Figure 5 evaluates this relationship over 20 episodes per task. For *move_stapler_pad*, 76 paired decisions yield a Spearman correlation of 0.719 (episode-bootstrap 95% CI: $[0.611, 0.799]$) between s_{1step} and the RMSE between the draft and oracle actions. For *put_bottles_dustbin*, 100 paired decisions yield a correlation of 0.779 ($[0.704, , 0.832]$). The one-step score also identifies oracle-rejected actions with AUROC 0.896 and 0.900, respectively. Overall, these results indicate that a single model evaluation provides a substantially informative signal of full-oracle disagreement.

A.3 Full Task Results on RoboTwin 2.0

This section reports the full per-task results on RoboTwin 2.0 in Table 4 and Table 5, complementing the representative examples shown in the main paper.

A.4 Comparison of our method with Sync and Async

Figure 6 compares synchronous inference, naive asynchronous inference, and SAVE-WAM. Synchronous inference uses the latest observation before generating each action chunk, providing fresh feedback but introducing inference stalls. Asynchronous inference overlaps generation with execution to reduce such stalls, but its actions are generated from older observations and therefore lack feedback from ongoing execution. SAVE-WAM retains asynchronous generation while introducing post-generation feedback: draft actions are verified using newer observations, accepted when they remain consistent, and replanned when a mismatch is detected. This provides a better balance between continuous execution and timely feedback.

A.5 Verification–Execution Pipelining

Ahead verification is designed to overlap verification with execution of the current action segment, thereby minimizing the additional waiting exposed at the boundary before the next segment can be executed.

Figure 7 contrasts these timing schemes. In standard asynchronous execution, the next action chunk is generated before the preceding actions finish, so the generation observation becomes stale relative to the state encountered at execution time. The generated actions are then executed without being reconsidered using newer observations. A direct alternative is to verify each segment immediately before execution using the latest observation. Although this provides the freshest feedback, the verification latency lies on the critical path and can interrupt streaming execution.

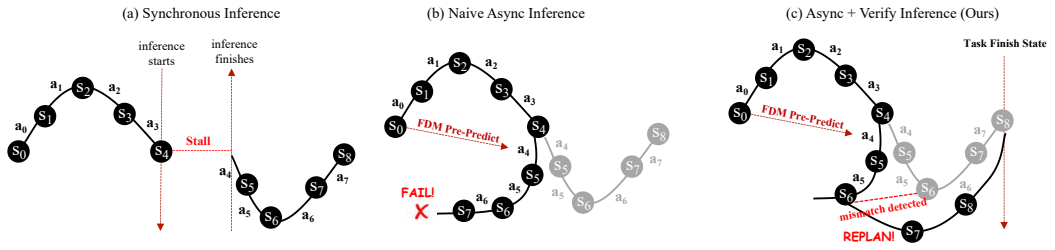


Figure 6: **Overview of feedback-gated asynchronous execution.** Synchronous inference provides fresh feedback but suffers from inference stalling. Naive asynchronous inference improves throughput by generating future actions ahead of time, but executes them under stale observations. SAVE-WAM closes this feedback gap by verifying speculative actions and selectively accepting valid prefixes while replanning upon mismatch.

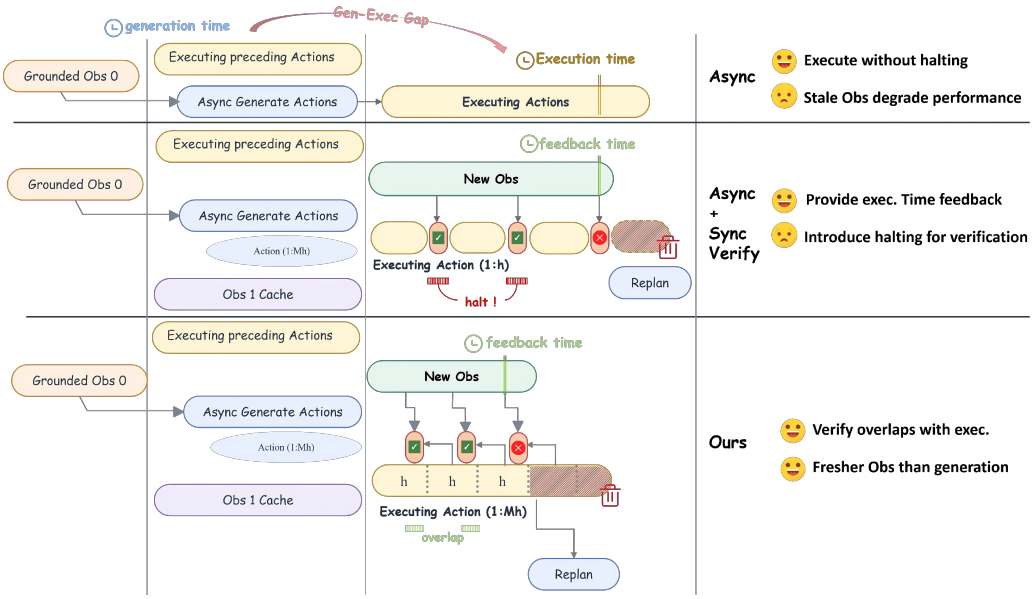


Figure 7: **Verification timing under asynchronous execution.** *Top:* Standard asynchronous inference generates actions from an earlier observation while preceding actions are still being executed, creating a generation–execution gap without post-generation feedback. *Middle:* Execution-time synchronous verification uses a fresh observation immediately before each segment, but places verification on the execution critical path and may introduce stalls. *Bottom:* SAVE-WAM moves verification ahead and overlaps it with execution of the current segment. Valid segments proceed continuously, whereas a detected mismatch rejects the remaining speculative suffix and triggers replanning.

SAVE-WAM instead exploits the substantially lower cost of verification relative to full action generation and performs it ahead of execution. While the robot executes the current h-step segment, a more recent observation is used to verify the upcoming speculative segment. If verification succeeds, the next segment is already available at the execution boundary; if it fails, the unexecuted suffix is discarded and replanning is triggered. This pipelined design therefore trades a small amount of observation freshness for substantially less exposed verification latency.

To quantify this effect, we analyze archived RoboTwin ahead-verification logs and measure the residual blocking time at each execution boundary. For a recorded verification event i, we define

$$W_i = t_i^{\text{return}} - t_i^{\text{collect}}, \tag{28}$$

where t_i^{collect} denotes the time at which the client requests the asynchronous verification result through `future.result()` after executing the current segment, and t_i^{return} denotes the time at which the call returns. Thus, W_i measures the *exposed boundary wait*, rather than the total verification computation time. For cross-draft events, this quantity may additionally include waiting for the main-model prefetch dependency.

As shown in Table 6, 54,801 of the 54,854 recorded events incur at most 1 ms of exposed waiting at the execution boundary. The pooled mean wait is only 0.931 ms, with a 90th percentile of 0.07 ms. The larger mean for cross-draft events, despite their small 90th-percentile value, is caused by a small tail of longer waits associated with cross-draft dependencies. Overall, these measurements show that verification is typically completed within the concurrent execution window, leaving negligible additional blocking on the execution path.

Table 4: Per-task results of LingBot-VA on RoboTwin. E and H denote the clean (easy) and random (hard) settings, respectively. Time is the successful episode average completion time of each task.

Task	Setting	Ours			FDM Async-32			FDM Async-64			Sync-16			AAC		
		SR ↑	Chunk	Time (s)	SR ↑	Chunk	Time (s)	SR ↑	Chunk	Time (s)	SR ↑	Chunk	Time (s)	SR ↑	Chunk	Time (s)
Adjust Bottle	E	98	29.98	56.873	96	32	49.510	94	64	46.894	**100**	16	59.762	94	25.89	52.215
	H	98	27.77		96	32		**100**	64		98	16		94	26.59	
Beat Block Hammer	E	**98**	28.22	53.645	90	32	47.031	92	64	45.052	96	16	55.941	80	25.38	49.135
	H	**100**	27.36		76	32		80	64		96	16		74	26.43	
Blocks Ranking RGB	E	96	36.05	181.772	90	32	158.996	86	64	154.400	**98**	16	198.278	94	31.27	170.806
	H	**100**	29.09		92	32		88	64		94	16		96	31.63	
Blocks Ranking Size	E	**96**	35.66	186.307	86	32	160.453	78	64	148.515	94	16	209.906	82	31.18	165.439
	H	**94**	29.05		82	32		60	64		92	16		68	31.01	
Click Alarmclock	E	**100**	28.50	31.215	**100**	32	28.963	**100**	64	26.414	**100**	16	31.704	**100**	23.59	31.662
	H	98	26.06		90	32		**100**	64		**100**	16		**100**	24.24	
Click Bell	E	**100**	23.71	30.055	**100**	32	26.923	**100**	64	25.088	**100**	16	30.929	**100**	27.32	27.192
	H	**100**	26.16		**100**	32		**100**	64		**100**	16		**100**	25.02	
Dump Bin Bigbin	E	**100**	32.28	84.250	92	32	79.491	88	64	65.127	92	16	89.037	94	29.02	73.592
	H	98	28.23		96	32		94	64		**100**	16		98	28.93	
Grab Roller	E	**100**	31.67	42.559	**100**	32	39.808	**100**	64	38.024	**100**	16	47.370	**100**	23.98	42.021
	H	**100**	27.86		96	32		**100**	64		**100**	16		**100**	24.60	
Handover Block	E	94	32.75	122.557	64	32	102.621	60	64	98.670	**100**	16	128.638	82	28.33	109.414
	H	72	29.42		66	32		62	64		**94**	16		62	28.45	
Handover Mic	E	96	33.64	98.739	**100**	32	86.346	92	64	81.747	98	16	105.132	78	30.72	86.541
	H	96	28.62		96	32		94	64		**98**	16		88	30.40	
Hanging Mug	E	**38**	35.03	138.995	26	32	128.688	4	64	121.281	30	16	156.605	20	30.65	135.278
	H	42	28.03		30	32		2	64		**46**	16		16	30.83	
Lift Pot	E	98	28.58	58.653	92	32	54.038	**100**	64	51.138	**100**	16	61.929	70	26.65	55.131
	H	**100**	27.62		96	32		**100**	64		96	16		86	27.24	
Move Can Pot	E	96	29.88	68.693	84	32	60.265	88	64	56.622	**98**	16	74.735	94	26.12	63.904
	H	90	28.14		86	32		**98**	64		82	16		80	26.45	
Move Pillbottle Pad	E	**100**	29.13	62.382	92	32	53.415	90	64	50.997	**100**	16	68.700	94	27.22	56.404
	H	94	28.43		90	32		88	64		**96**	16		84	26.48	
Move Playingcard Away	E	**100**	27.06	53.437	98	32	43.763	98	64	41.430	**100**	16	56.544	94	27.56	48.122
	H	**100**	27.05		98	32		98	64		94	16		94	29.17	
Move Stapler Pad	E	72	29.91	80.236	38	32	61.609	32	64	62.797	**76**	16	80.733	50	29.33	69.516
	H	**54**	27.88		44	32		42	64		**54**	16		52	27.84	
Open Laptop	E	**96**	34.86	72.125	**96**	32	68.166	90	64	68.016	94	16	88.180	**96**	24.54	79.815
	H	92	30.38		84	32		**100**	64		96	16		96	24.29	
Open Microwave	E	84	37.35	184.449	60	32	173.047	56	64	168.349	**94**	16	218.494	58	28.28	274.445
	H	72	33.50		56	32		40	64		**98**	16		50	29.28	
Pick Diverse Bottles	E	**98**	27.62	55.064	80	32	47.197	82	64	44.490	90	16	56.574	86	22.48	49.867
	H	80	27.32		**82**	32		80	64		78	16		80	22.36	
Pick Dual Bottles	E	**100**	27.82	54.979	88	32	47.857	80	64	44.269	**100**	16	56.161	**100**	22.61	50.671
	H	**94**	27.45		86	32		90	64		82	16		86	21.71	
Place A2B Left	E	94	28.76	64.933	88	32	55.967	88	64	53.842	**96**	16	70.661	90	27.13	63.221
	H	**96**	28.19		94	32		92	64		**96**	16		92	26.87	
Place A2B Right	E	92	31.28	62.502	**98**	32	56.781	88	64	54.225	94	16	68.782	84	26.13	58.280
	H	96	28.15		88	32		82	64		**98**	16		96	25.70	
Place Bread Basket	E	**98**	32.10	99.085	86	32	85.876	88	64	81.110	92	16	104.742	92	27.96	89.718
	H	94	28.08		86	32		84	64		**98**	16		80	27.03	
Place Bread Skillet	E	94	28.02	65.645	**96**	32	59.391	94	64	58.671	94	16	71.603	90	26.21	63.851
	H	**92**	28		86	32		88	64		86	16		82	24.81	
Place Burger Fries	E	96	34.64	115.095	**100**	32	97.388	96	64	90.321	96	16	120.452	96	27.36	98.655
	H	94	28.98		**100**	32		96	64		**100**	16		94	27.19	
Place Can Basket	E	82	34.70	123.526	74	32	113.822	76	64	105.451	**84**	16	133.699	70	30.19	123.936
	H	**88**	29.02		80	32		80	64		86	16		80	30.34	
Place Cans Plasticbox	E	**100**	34.71	116.024	**100**	32	100.785	92	64	96.310	**100**	16	122.469	**100**	29.12	103.337
	H	**100**	29.15		**100**	32		90	64		**100**	16		98	28.73	
Place Container Plate	E	**100**	33.41	64.026	**100**	32	55.998	98	64	53.141	**100**	16	67.778	**100**	27.08	58.056
	H	**98**	28.79		94	32		**98**	64		96	16		96	26.98	
Place Dual Shoes	E	92	30.55	116.915	88	32	103.244	80	64	100.766	**96**	16	124.156	86	28.45	105.115
	H	90	28.20		86	32		86	64		**92**	16		84	26.20	
Place Empty Cup	E	**100**	32.46	69.181	96	32	61.361	**100**	64	58.775	**100**	16	76.610	**100**	28.78	65.372
	H	**100**	27.76		**100**	32		**100**	64		**100**	16		98	27.33	
Place Fan	E	**94**	28.81	66.664	80	32	56.908	80	64	54.146	92	16	72.645	92	27.83	61.947
	H	86	28.64		78	32		80	64		**88**	16		86	26.92	
Place Mouse Pad	E	84	32.31	67.663	74	32	60.938	72	64	55.205	**94**	16	71.768	82	27.04	61.146
	H	88	28.34		76	32		70	64		**96**	16		78	28.31	
Place Object Basket	E	**86**	33.07	121.307	76	32	105.033	76	64	103.648	80	16	128.577	76	29.53	108.944
	H	**86**	28.64		80	32		70	64		78	16		64	31.61	
Place Object Scale	E	**100**	28.14	63.144	96	32	56.317	92	64	54.001	**100**	16	68.638	90	26.33	63.812
	H	**96**	28.21		88	32		88	64		94	16		84	25.78	
Place Object Stand	E	96	31.26	62.011	96	32	54.374	**98**	64	50.893	**98**	16	65.147	94	28.01	55.379
	H	94	28.60		**98**	32		**98**	64		**98**	16		92	27.35	
Place Phone Stand	E	**94**	26.04	54.567	74	32	48.666	70	64	47.915	92	16	60.312	90	30.28	52.258
	H	96	26.48		90	32		**98**	64		96	16		90	30.43	
Place Shoe	E	98	31.80	72.047	94	32	60.455	94	64	57.482	**100**	16	83.048	96	28.09	62.476
	H	96	28.35		94	32		**100**	64		96	16		96	27.82	
Press Stapler	E	86	31.49	50.360	**88**	32	49.261	86	64	51.812	82	16	54.852	86	22.77	53.762
	H	80	29.33		80	32		**96**	64		84	16		80	23.42	
Put Bottles Dustbin	E	80	32.72	275.897	62	32	235.294	46	64	233.738	**82**	16	273.918	58	31.53	248.468
	H	78	29.46		60	32		48	64		**80**	16		58	31.29	
Put Object Cabinet	E	74	29.81	140.887	74	32	119.691	**78**	64	112.570	76	16	144.997	58	29.68	123.123
	H	**82**	28.02		70	32		62	64		80	16		68	31.77	
Rotate QRcode	E	**98**	33.36	65.569	92	32	57.623	**98**	64	56.590	**98**	16	70.962	96	27.28	62.715
	H	90	28.33		82	32		94	64		**96**	16		86	25.60	
Scan Object	E	**88**	32.72	77.713	86	32	69.916	84	64	65.545	**88**	16	80.481	84	25.91	70.836
	H	90	29.11		86	32		**94**	64		88	16		68	25.21	
Shake Bottle	E	**100**	27.02	47.792	98	32	42.126	98	64	39.142	**100**	16	49.830	96	24.36	44.217
	H	92	27.77		94	32		96	64		**98**	16		94	24.47	
Shake Bottle Horizontally	E	98	32.85	46.919	98	32	41.995	**100**	64	39.047	**100**	16	50.218	**100**	24.31	43.873
	H	94	27.69		90	32		94	64		**96**	16		94	24.15	
Stack Blocks Three	E	**98**	36.41	184.893	60	32	157.519	54	64	149.612	**98**	16	197.193	78	31.40	167.722
	H	96	29.38		56	32		52	64		**98**	16		68	30.97	
Stack Blocks Two	E	**100**	36.10	121.614	**100**	32	105.551	**100**	64	99.140	98	16	132.231	**100**	30.99	110.059
	H	**100**	28.86		98	32		94	64		**100**	16		**100**	31.05	
Stack Bowls Three	E	80	37.85	206.929	82	32	185.361	68	64	182.426	**86**	16	216.308	76	30.83	187.774
	H	76	29.46		72	32		74	64		**84**	16		78	31.98	
Stack Bowls Two	E	**96**	36.64	129.979	90	32	115.667	92	64	107.338	94	16	139.929	90	30.03	119.606
	H	92	29.10		90	32		92	64		**98**	16		90	30.09	
Stamp Seal	E	**100**	29.97	59.814	78	32	54.773	76	64	49.932	98	16	64.284	90	27.15	63.290
	H	74	28.50		80	32		**84**	64		**84**	16		66	27.10	
Turn Switch	E	80	29.03	41.575	**86**	32	29.155	66	64	30.943	72	16	37.008	60	26.67	55.941
	H	52	28.27		**66**	32		64	64		56	16		60	26.40	
Average (50 tasks)	E	92.76	31.55	91.425	85.64	32.00	80.308	82.76	64.00	76.661	**92.80**	16.00	97.973	85.32	27.61	86.761
	H	89.20	28.41		83.68	32.00		83.20	64.00		**90.68**	16.00		82.08	27.52	

Table 5: Per-task results of FastWAM on RoboTwin. E and H denote the clean (easy) and random (hard) settings, respectively. Time is the task-average successful-episode latency.

Task	Setting	Ours			RTC			Sync-12			AAC		
		SR ↑	Chunk	Time (s)	SR ↑	Chunk	Time (s)	SR ↑	Chunk	Time (s)	SR ↑	Chunk	Time (s)
Adjust Bottle	E	**100**	19.29	27.010	**100**	18.77	24.380	**100**	12.00	39.660	**100**	17.23	28.930
	H	**100**	18.36		**100**	18.74		**100**	12.00		**100**	15.99	
Beat Block Hammer	E	**100**	17.94	28.250	98	17.83	22.290	98	12.00	38.840	98	16.85	28.000
	H	**100**	18.40		96	18.31		96	12.00		**100**	16.20	
Blocks Ranking RGB	E	**100**	19.74	69.840	92	19.77	62.430	**100**	12.00	75.840	**100**	16.34	84.240
	H	**100**	18.82		**100**	19.65		**100**	12.00		94	15.97	
Blocks Ranking Size	E	92	19.82	83.190	92	19.78	63.810	**96**	12.00	78.170	94	16.30	89.900
	H	**96**	18.78		90	19.77		94	12.00		**96**	16.06	
Click Alarmclock	E	**100**	16.06	17.580	**100**	18.60	14.080	**100**	12.00	31.430	**100**	16.48	17.300
	H	**100**	17.24		**100**	18.08		**100**	12.00		**100**	14.53	
Click Bell	E	**100**	17.66	16.570	**100**	18.22	13.570	**100**	12.00	31.250	**100**	17.58	16.820
	H	**100**	17.79		98	17.92		**100**	12.00		**100**	14.41	
Dump Bin Bigbin	E	**96**	18.84	36.450	**96**	19.08	33.630	**96**	12.00	50.580	92	16.21	45.200
	H	96	18.28		96	19.21		**98**	12.00		**98**	16.20	
Grab Roller	E	**100**	17.51	22.920	**100**	17.97	19.150	**100**	12.00	35.830	**100**	16.23	22.600
	H	**100**	16.79		**100**	17.17		**100**	12.00		**100**	15.54	
Handover Block	E	98	19.27	52.470	**100**	19.25	49.210	98	12.00	59.010	90	16.30	56.880
	H	**88**	18.70		78	19.65		82	12.00		86	16.14	
Handover Mic	E	98	18.88	48.700	**100**	19.13	38.800	98	12.00	55.720	**100**	16.30	49.250
	H	**100**	18.19		98	19.10		**100**	12.00		**100**	15.68	
Hanging Mug	E	58	19.52	63.670	54	19.62	61.630	**66**	12.00	68.860	56	16.18	68.130
	H	68	18.62		58	19.55		62	12.00		**70**	16.00	
Lift Pot	E	**100**	17.89	30.110	**100**	17.76	27.810	**100**	12.00	40.110	**100**	17.03	28.250
	H	**100**	17.32		**100**	17.55		**100**	12.00		**100**	15.14	
Move Can Pot	E	90	19.53	34.230	72	19.28	26.590	**96**	12.00	43.140	92	16.42	33.990
	H	94	18.26		82	19.02		**98**	12.00		92	15.78	
Move Pillbottle Pad	E	**100**	19.22	30.050	98	18.93	22.610	98	12.00	41.110	**100**	16.57	31.510
	H	98	18.35		98	19.01		96	12.00		**100**	15.96	
Move Playingcard Away	E	**100**	17.77	24.960	98	17.67	19.030	**100**	12.00	37.270	**100**	16.81	27.040
	H	98	18.07		**100**	17.52		**100**	12.00		**100**	16.46	
Move Stapler Pad	E	**90**	18.73	34.020	78	19.25	28.090	76	12.00	42.810	86	16.12	34.360
	H	**78**	18.40		76	19.24		74	12.00		68	15.94	
Open Laptop	E	98	18.03	33.260	98	17.95	26.690	98	12.00	43.330	**100**	16.10	31.360
	H	**100**	17.47		**100**	17.77		**100**	12.00		**100**	15.53	
Open Microwave	E	82	19.54	87.690	**88**	19.52	65.290	80	12.00	65.840	80	16.09	86.590
	H	58	18.89		48	19.86		58	12.00		**62**	15.93	
Pick Diverse Bottles	E	78	18.99	26.860	**82**	19.19	21.700	78	12.00	38.830	74	16.29	29.260
	H	84	18.11		88	18.89		88	12.00		**90**	16.05	
Pick Dual Bottles	E	98	18.28	24.610	**100**	18.50	20.730	**100**	12.00	38.930	98	16.46	27.200
	H	**100**	17.47		94	18.84		96	12.00		98	15.91	
Place A2B Left	E	**96**	19.41	31.200	94	19.20	23.810	94	12.00	41.610	**96**	16.90	31.510
	H	96	18.56		94	19.14		88	12.00		**98**	15.86	
Place A2B Right	E	**98**	19.33	30.900	94	19.33	25.030	96	12.00	41.900	**98**	16.61	32.450
	H	92	18.57		94	19.16		**98**	12.00		96	15.85	
Place Bread Basket	E	**94**	19.49	46.680	92	19.44	38.500	90	12.00	52.440	90	16.08	46.670
	H	96	18.25		92	19.39		**98**	12.00		92	15.92	
Place Bread Skillet	E	92	18.96	34.920	92	18.80	27.800	**94**	12.00	44.540	90	16.51	35.210
	H	**96**	18.04		**96**	18.75		86	12.00		**96**	16.08	
Place Burger Fries	E	96	19.08	50.860	94	19.15	41.630	**98**	12.00	56.620	**98**	16.40	51.020
	H	**100**	18.29		96	19.16		**100**	12.00		**100**	15.86	
Place Can Basket	E	68	19.41	63.970	68	19.22	57.410	**70**	12.00	65.360	60	16.14	63.410
	H	74	18.57		**80**	19.25		64	12.00		56	15.89	
Place Cans Plasticbox	E	**100**	18.79	54.670	98	18.49	45.110	96	12.00	58.530	94	16.17	53.250
	H	**100**	18.31		98	18.71		**100**	12.00		98	15.94	
Place Container Plate	E	96	19.70	23.010	**100**	19.16	24.090	**100**	12.00	41.370	98	17.35	31.020
	H	98	18.69		98	18.99		**100**	12.00		**100**	15.70	
Place Dual Shoes	E	92	19.50	55.080	**94**	19.32	44.750	78	12.00	59.020	90	16.31	55.910
	H	**86**	18.55		**86**	19.37		84	12.00		82	15.95	
Place Empty Cup	E	**100**	18.87	32.560	**100**	18.49	26.040	**100**	12.00	43.280	**100**	16.84	32.990
	H	**100**	18.51		98	18.60		**100**	12.00		**100**	16.31	
Place Fan	E	**98**	18.95	25.510	92	19.13	27.780	92	12.00	44.680	96	16.63	35.390
	H	96	18.39		**98**	18.89		94	12.00		**98**	16.18	
Place Mouse Pad	E	86	19.24	31.730	86	19.16	26.200	84	12.00	41.850	**88**	16.60	33.430
	H	**92**	18.54		82	19.22		84	12.00		88	15.83	
Place Object Basket	E	**90**	19.33	63.690	76	19.56	53.620	88	12.00	68.640	82	16.25	65.740
	H	80	18.67		82	19.58		78	12.00		**86**	15.91	
Place Object Scale	E	94	19.44	30.720	**96**	19.13	24.920	**96**	12.00	41.090	90	16.80	30.020
	H	**98**	18.29		90	19.22		92	12.00		96	15.95	
Place Object Stand	E	92	18.90	28.300	88	18.76	23.310	**96**	12.00	40.500	**96**	17.14	29.570
	H	90	18.73		**96**	18.82		94	12.00		90	15.75	
Place Phone Stand	E	**98**	19.11	27.240	96	19.43	22.000	94	12.00	38.400	96	16.16	27.270
	H	96	17.75		**100**	19.31		94	12.00		98	15.82	
Place Shoe	E	94	18.79	34.580	94	19.36	31.090	**96**	12.00	44.210	**96**	16.18	34.480
	H	**98**	18.08		96	19.40		**98**	12.00		**98**	15.98	
Press Stapler	E	96	17.35	22.320	96	17.37	20.180	**98**	12.00	35.820	**98**	16.23	22.200
	H	**100**	16.87		**100**	16.80		**100**	12.00		**100**	15.23	
Put Bottles Dustbin	E	86	19.88	83.050	84	19.78	75.650	**90**	12.00	92.190	84	16.11	106.360
	H	**88**	18.78		78	19.84		86	12.00		86	16.09	
Put Object Cabinet	E	**94**	19.48	56.200	90	19.50	52.270	88	12.00	63.850	90	16.55	60.860
	H	**92**	18.61		80	19.64		86	12.00		90	15.95	
Rotate QRcode	E	90	19.48	25.200	**92**	19.09	27.340	90	12.00	43.380	90	16.55	35.670
	H	90	18.46		**94**	19.18		92	12.00		84	15.90	
Scan Object	E	**98**	19.23	38.380	92	19.27	35.870	92	12.00	50.620	94	16.53	40.880
	H	90	18.47		92	19.23		**98**	12.00		92	16.15	
Shake Bottle	E	**100**	18.40	22.870	**100**	18.30	17.390	**100**	12.00	36.170	**100**	16.85	24.300
	H	**100**	17.76		**100**	18.14		**100**	12.00		**100**	16.70	
Shake Bottle Horizontally	E	**100**	18.35	22.900	**100**	18.21	19.200	**100**	12.00	36.100	**100**	16.87	22.360
	H	**100**	17.79		**100**	18.10		**100**	12.00		**100**	16.63	
Stack Blocks Three	E	96	19.68	81.210	**100**	19.63	71.350	**100**	12.00	76.600	**100**	16.33	86.220
	H	94	18.81		98	19.62		**100**	12.00		98	16.00	
Stack Blocks Two	E	**100**	19.47	55.460	**100**	19.31	43.390	**100**	12.00	58.820	**100**	16.33	57.960
	H	**100**	18.68		98	19.40		**100**	12.00		**100**	16.05	
Stack Bowls Three	E	76	19.88	80.870	66	19.79	78.730	76	12.00	83.230	**78**	16.28	93.050
	H	**84**	18.93		**84**	19.75		80	12.00		62	16.07	
Stack Bowls Two	E	86	19.82	56.840	84	19.51	45.320	92	12.00	60.490	**94**	16.49	62.870
	H	**98**	18.79		**98**	19.39		**98**	12.00		92	15.95	
Stamp Seal	E	82	19.18	28.420	**88**	18.99	22.080	**88**	12.00	39.670	**88**	16.76	30.640
	H	92	18.49		88	18.87		**94**	12.00		**94**	15.80	
Turn Switch	E	**60**	19.26	19.430	54	19.14	15.290	58	12.00	31.730	52	16.30	16.400
	H	62	18.64		62	19.37		**66**	12.00		64	15.84	
Average (50 tasks)	E	**92.52**	18.97	41.024	90.92	18.98	34.973	92.24	12.00	49.785	91.72	16.50	43.718
	H	**92.72**	18.28		90.96	18.94		91.88	12.00		91.76	15.89	

Table 6: Boundary-wait diagnostics for ahead verification with $k = 16$. The window is the mean time from asynchronous submission to result collection, including concurrent execution and any residual waiting. All recorded events are included regardless of task success. The final column reports the fraction of events with at most 1 ms exposed wait and should not be interpreted as a compute-overlap ratio.

Event type	Events	Window (s)	Mean W (ms)	P90 W (ms)	$W \leq 1$ ms (%)
Intra-draft	37,697	5.883	0.042	0.09	99.997
Cross-draft	17,157	5.504	2.884	0.05	99.697
Pooled	54,854	5.765	0.931	0.07	99.903