SPEC-WAM: FEEDBACK-GROUNDED SPECULATIVE INFERENCE FOR WORLD-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

World-action models (WAMs) leverage video-generation priors for robot action generation, but iterative denoising delays their response to new observations. Computing ahead during action execution can reduce this delay, yet predicted video and actions may become inconsistent with the observations that actually arrive. We introduce Spec-WAM, a training-free framework that tightly couples speculation across diffusion denoising and control loops. More specifically, during denoising sampling, extrapolation provides candidate action-denoising states for batched verification, amortizing model evaluation overhead. Within the control loop, the same pretrained WAM autoregressively prepares the next video-action trajectory while the robot executes its current action. After receiving the new observation, Spec-WAM checks whether the drafted video can be reused for action generation. For models that generate video before actions, it can reuse the accepted video together with its paired draft actions. For models that jointly denoise video and actions, a shared model query checks both streams, and action denoising continues from the last state that passes verification instead of restarting from noise. On RoboTwin 2.0 with unseen instructions, Spec-WAM matches the task success of OpenWAM's released inference on paired episodes (92.49% versus 92.65%) while significantly reducing mean feedback-to-action latency by 21.7%. On Next Forcing, it keeps task success and lowers this latency 34-fold, including the effect of fewer video denoising steps and of batched checking, which cuts sequential action-model calls from 51 to 4.02.

1 INTRODUCTION

World–action models (WAMs) transfer video-generation priors to robot control by learning visual dynamics together with action generation (Li et al., 2026; Xu et al., 2026; Wang et al., 2026). Predicted scene evolution gives the policy visual context for choosing how the robot should move. At each decision, the model outputs an *action chunk*, a short sequence of robot actions, and the observations collected during execution provide feedback for the next decision. Producing a chunk requires many sequential network evaluations to denoise video and action states, whether video is generated before actions or both are denoised jointly. Because this computation follows every new observation, it delays the next action conditioned on that observation; when the scene changes during execution, the delay postpones the robot's response to the change. We therefore target *feedback-to-action latency*, the interval from submitting feedback to receiving the next action chunk, and evaluate its trade-off with task success.

Recent work reduces WAM inference cost in several ways. Fast-WAM omits future-video generation at inference after video co-training (Yuan et al., 2026), and Faster-WAM computes future representations once and selectively exposes them to action denoising (Zhao et al., 2026); both build reduced future prediction into the learned policy. Caching and runtime optimization reduce repeated computation and execution overhead (Liu et al., 2025; Ma et al., 2025). Asynchronous execution overlaps inference with action execution (Black et al., 2025), but a chunk conditioned on a new observation still waits for the full inference computation after it. For a pretrained WAM that retains future prediction, we ask: *can computation prepared during action execution shorten the response to the next observation?* Such computation starts from predicted feedback that may not match the real observation, so it is useful only if checking it and completing the accepted part take less time than generating the decision after the observation.

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.

Two sequential dependencies in WAM inference determine where such computation can be prepared and where it can be checked (Figure 2). Along the **denoising axis**, indexed by i, each model evaluation within a decision takes the state produced by the preceding denoising step as input. As in speculative decoding and parallel sampling (Leviathan et al., 2023; Shih et al., 2023), candidate states can be proposed without network evaluation and checked together by the pretrained model, so that an accepted prefix advances several denoising steps in one sequential round. Checking still evaluates the network on every candidate state; batching reduces its cost when shared computation and memory traffic can be amortized (Williams et al., 2008), a condition that differs between action-only queries and joint queries that also carry video states (Section D). Along the **execution axis**, indexed by control chunk t, the next decision depends on feedback produced while the current chunk executes. This execution window leaves time to prepare the next video–action trajectory from predicted feedback, which is useful only if it remains compatible with the observation that actually arrives. The execution axis therefore supplies computation performed before feedback, and the denoising axis reduces the cost of checking and completing it afterwards.

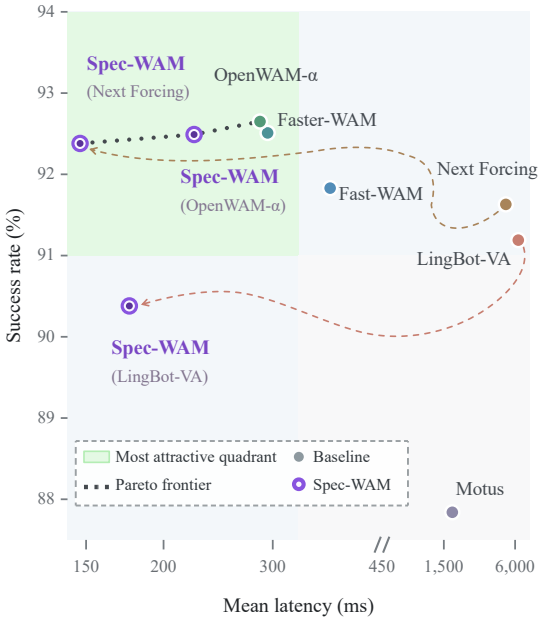

Figure 1: **Success–latency trade-off on RoboTwin 2.0.** Points show average success and mean latency from Table 2. Dots trace the Pareto frontier; dashed arrows link baselines to their Spec-WAM variants. Green marks the lower-latency, higher-success region. The log latency axis is compressed above 450 ms.

We introduce **Spec-WAM**, a training-free framework that follows one principle: *compute ahead; commit after feedback*. While the robot executes the current action chunk, the same pretrained WAM autoregressively drafts the next video–action trajectory from predicted feedback. When the real observation arrives, the model evaluates the saved trajectory under the updated conditioning: video that passes this check is reused, and action generation continues from the last accepted denoising state instead of starting from noise. Rejected or unfinished candidates are replaced by recovery, which generates the next chunk under the real observation. Within each decision, an extrapolation-based drafter proposes later action-denoising states from model outputs already computed, and a single batched evaluation checks them, so checking and continuation advance several denoising steps per sequential call.

Reusing prepared video and actions must respect how each model couples the two streams. LingBot-VA and Next Forcing first predict future video and then generate actions conditioned on it (Li et al., 2026; Xu et al., 2026); once a video is selected, it fixes the condition under which saved action states are checked and continued. OpenWAM-α instead denoises video and actions jointly, with each stream attending to the other (Wang et al., 2026); an action state can therefore be evaluated only together with its paired video state and timesteps. Spec-WAM applies the same **draft–verify–reuse** procedure to both, forming every check as a complete query of the pretrained model. When a video check and an action check require identical inputs, one model evaluation serves both, and its output is reused for continuation. Spec-WAM thus preserves each model's video–action dependencies, not its exact sampling trajectory.

Experiments on RoboTwin 2.0 examine the complete framework before separating its mechanisms. With unseen instructions, Spec-WAM on OpenWAM-α matches the task success of the released inference program on paired episodes (92.49% versus 92.65%) and lowers mean latency from 286.13 to 224 ms, using a second GPU for recovery. The action-solver study on Next Forcing independently reduces mean sequential target calls from 51 to 4.02 on the original action grid. Cross-model gate comparisons and deployment studies expose how selection, query cost, and the available execution window shape the benefit. These observations motivate evaluating complete quality–latency oper-

ating points and then testing the contribution of each component. To summarize, our contributions are:

- **Dual-Axis Speculation.** We combine batched evaluation of analytic denoising candidates with autoregressive self-drafting during action execution.
- **Feedback-Grounded Verification.** We verify candidate video and action trajectories under latest real observations, reuse video predictions that pass verification, and resume action generation from accepted denoising states.
- **Cross-Architecture Acceleration Framework.** We develop a training-free framework that supports both video-conditioned inverse-dynamics models and joint video–action diffusion models.

2 Related Work

World-action models. WAMs connect visual dynamics modeling with robot action generation. Cosmos Policy incorporates actions into a video diffusion process, while DreamZero jointly generates future video and actions (Kim et al., 2026; Ye et al., 2026). LingBot-VA develops causal video-action predication, and Next Forcing extends its prediction objective to multiple future chunks (Li et al., 2026; Xu et al., 2026). OpenWAM systematically examines architecture, video-action information flow, and denoising schedules (Wang et al., 2026). Recent efficiency-oriented designs modify how future representations are used at inference. Fast-WAM retains video co-training but omits future-video generation at inference, whereas Faster-WAM computes future representations once and selectively reuses them during action denoising (Yuan et al., 2026; Zhao et al., 2026). Spec-WAM accelerates pretrained WAMs without additional training while retaining future-video prediction. It prepares video–action trajectories during robot execution and verifies them under new observations, reusing accepted video predictions and action-denoising states.

Speculative inference and parallel denoising. Speculative decoding verifies proposed tokens through batched target evaluations (Leviathan et al., 2023; Chen et al., 2025b). For diffusion models, ParaDiGMS and ParaTAA parallelize sampling through iterative refinement of candidate trajectories (Shih et al., 2023; Tang et al., 2024), while stochastic speculative diffusion uses transition-based acceptance and correction to preserve target distributions (De Bortoli et al., 2025; Soen et al., 2026). In action generation, Spec-VLA relaxes token acceptance using action-space distances (Wang et al., 2025), and FLASH trains a lightweight action drafter and checks its proposals at selected denoising times to select an executable action prefix (Niu et al., 2026). Spec-WAM instead accepts a prefix of the denoising trajectory for an entire action chunk and resumes denoising from the last accepted state. Its candidate states are constructed from previously computed model outputs and verified in a batch, without a separately trained drafter.

Asynchronous control and efficient execution. Asynchronous inference overlaps action generation with robot execution. Real-time chunking uses inpainting to keep the new chunk consistent with the actions that will execute while it is being generated (Black et al., 2025). Predictive WAMs also exploit the execution window to prepare subsequent decisions (Team, 2026; Zhang et al., 2026). DexWorldModel performs early denoising under predicted visual features while the robot executes its current action (Deng et al., 2026). When the next observation arrives, it updates the conditioning and resumes denoising from a preset intermediate flow time. Spec-WAM makes reuse conditional on explicit verification of the prepared video and action trajectories under the new observation. Video checks determine whether to retain the prepared predictions, while action checks select the denoising prefix from which generation can resume.

3 Preliminaries

WAM inference. Let t index control decisions and i index denoising states within one decision. At decision t, a pretrained WAM with parameters θ generates an action chunk u_t from the observed history H_t, the camera observations and executed actions available at that decision; task conditioning is left implicit. At denoising index i, z_i is the video latent and x_i the normalized state of the entire action chunk, whose final value gives u_t. The two streams have noise levels σ_i^v and σ_i^a, which need not coincide and which the model receives as timesteps s_i^v and s_i^a.

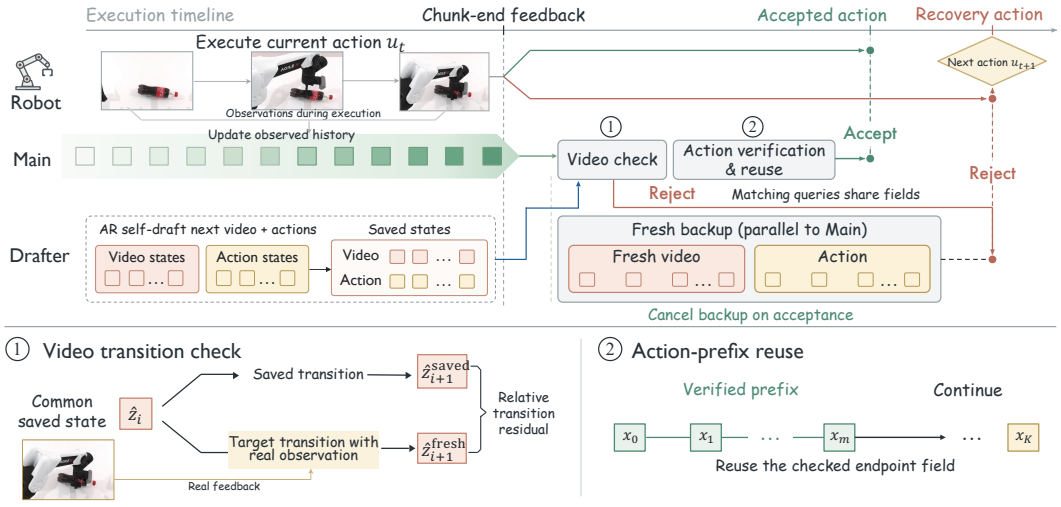

Figure 2: **Spec-WAM pipeline.** Main and Drafter are replicas of one pretrained WAM on two GPUs. During execution, Drafter prepares the autoregressive draft. When feedback arrives, Main checks the draft's video (1) and action prefix (2) (Section 4.3) while Drafter runs recovery, which acceptance cancels. Models that generate actions from video skip (2) after accepting the video.

Feedback-to-action latency. Executing u_t yields new observations, the feedback that establishes H_{t+1}. We define feedback-to-action latency, or simply *latency*, as the time from submitting this feedback to receiving u_{t+1}. It includes the work that remains after submission: encoding the new observations and committed actions into the model's history, and generating u_{t+1}. It excludes sensor capture, actuation, and the execution window of u_t, so computation completed while u_t executes does not count toward latency.

Batched verification. Checking candidate denoising states requires one network evaluation per state, but these evaluations are mutually independent and can be issued as one batched query. Let a query of k states perform C_k operations and transfer Q_k bytes. For compute throughput Π and memory bandwidth β, the roofline model bounds its execution time T_k (Williams et al., 2008):

$$T_k \geq \max\left\{\frac{C_k}{\Pi}, \frac{Q_k}{\beta}\right\}. \tag{1}$$

Model weights and conditioning shared across the batch can be read once per query, whereas operations grow with every added state, so the arithmetic intensity $I_k = C_k/Q_k$ increases with k. Below the ridge point $I_k = \Pi/\beta$, memory traffic bounds T_k and additional states add little time; above it, computation does, and each state adds proportional cost. Batching also amortizes per-query overheads outside this bound, such as kernel dispatch (Wang et al., 2020).

4 Spec-WAM

Spec-WAM computes candidate denoising states before the model evaluations that would ordinarily produce them, and keeps a candidate only after the pretrained model checks it (Figure 2). Candidates come from analytic drafts within a decision (Section 4.1) and autoregressive drafts prepared while the current chunk executes (Section 4.2). Both are checked against the model's own solver update under the observed history, which decides which states are kept and where the solver resumes (Section 4.3). Section 4.4 describes what each model's video–action coupling lets these checks share and reuse, and how a second replica runs recovery alongside them.

4.1 Denoising-axis speculation

Within a decision, each denoising state represents the entire action chunk, and the solver needs the model's velocity at one state to compute the next, so S solver steps make S dependent model

queries. We draft several next states without model evaluation and check them in one batched query; the noise grid is unchanged, and progress comes from accepting several transitions per query.

Analytic drafts. Under fixed conditioning, let $v^*(x, \sigma)$ be the model's action velocity on the grid $\sigma_0 > \cdots > \sigma_S = 0$, with $h_i = \sigma_{i+1} - \sigma_i$ and Euler steps $x_{i+1} = x_i + h_i v^*(x_i, \sigma_i)$. On the linear flow-matching path $x_\sigma = (1-\sigma)x_{\text{clean}} + \sigma\xi$, the velocity is $\xi - x_{\text{clean}}$, so each state with an evaluated velocity, (σ_a, x_a, v_a^*), estimates the clean chunk as $\bar{x}_a = x_a - \sigma_a v_a^*$. As in self-speculative diffusion (De Bortoli et al., 2025; Soen et al., 2026), the draft is built from outputs the model has already computed: from the latest two such states, $a < b$, it extrapolates this estimate and converts it back to a velocity:

$$\widehat{\bar{x}}(\sigma) = \begin{cases} \bar{x}_b + \dfrac{\sigma - \sigma_b}{\sigma_b - \sigma_a}(\bar{x}_b - \bar{x}_a), & \sigma \geq \sigma_{\text{hold}}, \\ \bar{x}_b, & \sigma < \sigma_{\text{hold}}, \end{cases} \qquad \widehat{v}(x, \sigma) = \frac{x - \widehat{\bar{x}}(\sigma)}{\max(\sigma, \epsilon_{\text{num}})}. \tag{2}$$

With one such state, its estimate is held throughout. From the current state $y_0 = x_j$, the draft forms $y_{r+1} = y_r + h_{j+r}\widehat{v}(y_r, \sigma_{j+r})$ over a window of $k = \min(K, S - 1 - j)$ transitions without model evaluation. We use $\sigma_{\text{hold}} = 0.33$ and $\epsilon_{\text{num}} = 10^{-4}$.

Why drafts can be accepted. The drafted velocity has the model's form, an offset from a clean estimate divided by σ. At a drafted state y_r with $\sigma = \sigma_{j+r} \geq \epsilon_{\text{num}}$ and $h = h_{j+r}$, let $\bar{x}_r^* = y_r - \sigma v^*(y_r, \sigma)$ be the model's clean estimate and $\widehat{\bar{x}}_r = \widehat{\bar{x}}(\sigma)$ the drafted one; then

$$y_{r+1} - \big(y_r + h\, v^*(y_r, \sigma)\big) = \frac{h}{\sigma}\big(\bar{x}_r^* - \widehat{\bar{x}}_r\big). \tag{3}$$

Relative to the model's own step, $(h/\sigma)(y_r - \bar{x}_r^*)$, this defect is the error of the drafted clean estimate divided by $\|y_r - \bar{x}_r^*\|$. Long drafts therefore pass when the model's clean estimate changes little or nearly linearly along the trajectory, which extrapolation follows at high noise. Because the denominator shrinks with σ, the same error weighs more at low noise; below σ_{hold} the draft therefore holds $\bar{x}_b$ and drops the extrapolated slope, whose error would weigh most there.

Batched checking. All drafted states are known before any is queried, so one batched query evaluates $v^*(y_r, \sigma_{j+r})$ for $r = 1, \ldots, k$. With the known velocity at y_0, the outputs give the model's step from every drafted state, which Section 4.3 compares with the drafted step. The same batched query also evaluates the saved states of an autoregressive draft (Section 4.2). A round that accepts m transitions replaces about mT_1 of sequential queries with T_k, so fewer sequential queries lower latency only if $T_k < mT_1$. Two properties set this margin: how predictable the clean estimate is, which determines m, and how slowly query time grows with k (Section 3), which determines T_k/T_1. Table 1 shows a wide margin for the action-only solver and a smaller one for the joint query.

Table 1: Batch cost. k: states per batched query; T_k/T_1: its time relative to a one-state query; Queries: sequential queries per decision, native → Spec-WAM.

	k	T_k/T_1	Queries
Next Forcing	≤49	1.52–2.00	51→4.02
OpenWAM-α	10	1.25	4→1

4.2 Autoregressive self-drafting

Along the execution axis, decision $t{+}1$ is conditioned on the feedback produced while u_t executes, so native inference can start only after that feedback arrives. While u_t executes, the same pretrained WAM instead extends a predicted history by one autoregressive chunk, the autoregressive draft:

$$\begin{aligned} \mathcal{T}_{t+1} &= \text{Draft}_\theta(\widetilde{H}_{t+1}), \\ H_{t+1} &= \text{Ground}(H_t, u_t, \mathcal{O}_{t+1}). \end{aligned} \tag{4}$$

The first line runs during execution on the predicted history $\widetilde{H}_{t+1}$, in which the model's predicted video stands in for the observations of executing u_t; the second adds the actual observation $\mathcal{O}_{t+1}$ when feedback arrives. Asynchronous execution also computes during the execution window, but conditions the next chunk on the observations available when that computation starts (Black et al., 2025); the draft is instead conditioned on predicted feedback and is used only after the actual feedback has checked it. It saves its video and action denoising states, the model outputs needed for

checking or continuation, and the executed action it assumes. Checks query a context built from H_{t+1}, while the predicted history and the draft's caches stay private, so a check never conditions on the prediction it evaluates, which can diverge from what execution produces (Figure 3).

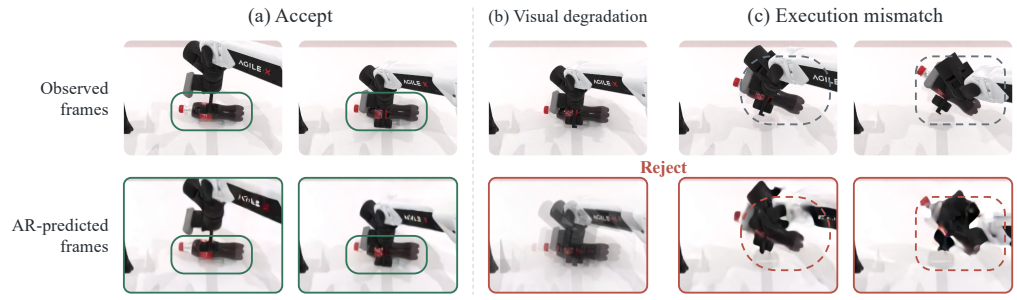

Figure 3: **Autoregressive drafts can diverge from execution.** Next Forcing examples show (a) an accepted draft that agrees with later observations and (c) a rejected draft compared with observations after recovery.

Drafting evaluates the model within the execution window, which does not count toward latency (Section 3). A draft is usable only if it is finished when feedback arrives and assumes the action that was actually published; otherwise recovery generates the next chunk under the observed history (Section 4.4). The next draft must extend a predicted history built on the real observation and the published action, not on predictions that feedback has not corrected. An accepted draft was computed on predicted history, so after acceptance a background pass, reanchoring, builds the next draft's history from them within the new execution window.

4.3 Feedback-Grounded Verification

A drafted transition can differ from the model's own update for two reasons: an analytic draft approximates the model's velocity, and an autoregressive draft was computed under predicted rather than observed history. One residual measures both. Let $\Phi_i(y; c)$ be one explicit Euler update of the model's solver from state y under condition c, where c_{pred} is the condition used to build the draft and c_{real} the one given by the observed history; for a joint model, y contains both streams. A drafted transition from $\widehat{y}_i$ to $\widehat{y}_{i+1}$ has residual

$$\begin{aligned} r_i &= \widehat{y}_{i+1} - \Phi_i(\widehat{y}_i; c_{\text{real}}) \\ &= \underbrace{\widehat{y}_{i+1} - \Phi_i(\widehat{y}_i; c_{\text{pred}})}_{\text{candidate numerical defect}} + \underbrace{\Phi_i(\widehat{y}_i; c_{\text{pred}}) - \Phi_i(\widehat{y}_i; c_{\text{real}})}_{\text{change in conditioning}}. \end{aligned} \tag{5}$$

For an analytic draft, $c_{\text{pred}} = c_{\text{real}}$ and only the defect of Equation 3 remains. For an autoregressive draft, the second term is the change that the new observation causes in the model's update, so even a draft that follows its predicted condition closely can fail the check. Verification computes r_i from the saved states and one query under c_{real}; the decomposition explains its sources and is not computed. The video check decides whether the saved video can condition action generation, and the action check selects the state from which the action solver continues.

Video-trajectory verification. The video check compares a saved video update with the update the model computes from the same state under the observed history. Let F_v be the model's video velocity and $h_i^v = \sigma_{i+1}^v - \sigma_i^v$. For the saved update $d_i = \widehat{z}_{i+1} - \widehat{z}_i$ (Figure 2(1)), the video part of r_i is $r_i^v = d_i - h_i^v F_v(\widehat{z}_i; c_{\text{real}})$. The residual-ratio gate scores a single transition $i^\star$, the saved state in the verification query whose noise level is closest to σ_{probe}, and accepts the drafted video if

$$g_v = \frac{\text{RMS}(r_{i^\star}^v)}{\text{RMS}(d_{i^\star})} \le \tau_v, \tag{6}$$

with RMS over all latent coordinates. The ratio measures the model's correction relative to the saved update: with $\tau_v = 0.2$, a draft passes if, under the observed history, the model's update differs from the saved one by at most 20% of the saved update's RMS. We use $\sigma_{\text{probe}} = 0.95$ and $\tau_v = 0.2$,

except $\tau_v = 0.25$ for LingBot-VA in Table 2. Because one transition decides for the whole video, this gate selects drafts rather than testing every transition as the action check below does; Figure 4 measures its effect on success.

Action-prefix verification and continuation. Let F_a be the model's action velocity, c contain the observed history and the selected video, and $h_i^a = \sigma_{i+1}^a - \sigma_i^a$. The action part of r_i and the length of the model's step are

$$e_i = \widehat{x}_{i+1} - \big(\widehat{x}_i + h_i^a F_a(\widehat{x}_i; c)\big), \qquad \ell_i = \mathrm{RMS}\big(h_i^a F_a(\widehat{x}_i; c)\big). \tag{7}$$

From the window's starting index j, with accumulated defect D and path length L before it, we accept the longest prefix m for which every $q \le m$ satisfies

$$D + \sum_{r=0}^{q-1} \mathrm{RMS}(e_{j+r}) \le \tau_a \left(L + \sum_{r=0}^{q-1} \ell_{j+r} \right). \tag{8}$$

with RMS over active normalized action coordinates and $\tau_a = 0.1$ unless stated otherwise. For an analytic draft, $\mathrm{RMS}(e_i)/\ell_i$ is the relative error of Equation 3, so the budget bounds its average over the committed path, weighted by the step lengths ℓ_i. Accepted transitions update D, L, and the solver resumes from the last accepted state (Figure 2(2)) with the velocity already evaluated there. If no transition is accepted, the solver takes one Euler step with the velocity it holds and queries the model at the new state, so every round advances at least one grid step. Accepted states thus lie on the model's grid, and every remaining step uses the model's own solver update. Unlike speculative sampling for diffusion models (De Bortoli et al., 2025; Soen et al., 2026), the test preserves neither the model's distribution nor the dense solution; it bounds accumulated local defects, and task success is evaluated empirically (Section 5). After an autoregressive draft's video is accepted, the same test applies to its saved action states under c, and continuation carries D, L from the accepted prefix; Section 4.4 describes which couplings run this check and which return the drafted action directly.

4.4 Dependency-aware execution

After feedback, an accepted draft supplies the next action, and an unusable or rejected one is replaced by recovery. What either branch can take from the draft depends on how the model couples video and actions. A saved state can be evaluated again under new conditioning, but a model output computed at that state can be reused only when the complete query agrees: conditioning, video and action states, timesteps, guidance, and cache context. In the two couplings we evaluate, this rule decides which checks share a query and what recovery can reuse. A model adapter applies it for each model: it rebuilds the conditioning from the observed history, evaluates the model at saved states without modifying the model's caches, and checks a draft's noise grid, initial noise, and conditioning identity before the draft is used.

Table 2: **Success–latency comparison on RoboTwin 2.0.** Success rate (%) in clean and randomized settings and mean latency (ms); † marks published success rates, and we run the other rows with unseen instructions. Small numbers give the change in average success (pp) of Spec-WAM over native inference of the same model. Green marks the top three per column.

Method	Clean	Rand.	Avg.	Lat. (ms)
Fast-WAM†	91.88	91.78	91.83	371.26
Faster-WAM†	92.78	92.26	92.52	290.82
Motus†	88.66	87.02	87.84	1,769.24
LingBot-VA	91.63	90.71	91.19	6,383.76
+ Spec-WAM	90.67	90.07	90.38 −0.81	176.14
Next Forcing	93.50	89.75	91.63	5,009.50
+ Spec-WAM	93.33	91.43	92.38 +0.75	146.68
OpenWAM-α	94.66	90.65	92.65	286.13
+ Spec-WAM	93.82	91.15	92.49 −0.17	224.00

Video-conditioned action generation. In LingBot-VA and Next Forcing, actions are generated after the video and conditioned on it (Li et al., 2026; Xu et al., 2026), so a change of video invalidates earlier action outputs but not the saved action states. After acceptance, checking the drafted actions would add a sequential pass of action queries to every accepted decision. We instead return the action drafted with the accepted video, which keeps the video-to-action dependency but leaves the action's direct dependence on the new observation unchecked; Appendix D measures what each choice costs. After rejection, recovery regenerates the video, re-evaluates a few saved action states under it, keeps the saved prefix that passes, and solves only the remaining steps.

Joint video–action denoising. In OpenWAM-α, video and actions are denoised jointly (Wang et al., 2026): each query takes both streams and their timesteps and returns both velocities, $(v^v, v^a) = F_\theta(z, x, s^v, s^a; H_{t+1})$. The video probe and the action check (Equation 8) query paired states of the same saved trajectory, so one batched query over their union returns both the video score and the action-prefix velocities. The action check thus adds states to an existing query instead of a sequential pass, and we always perform it. After acceptance, continuation pairs each new action state with the saved video state and timestep at the same index; after rejection, recovery runs the released joint schedule, which regenerates video and actions together.

Scheduling on two replicas. Run in sequence, a rejected draft would cost its check plus recovery. The second replica, which prepares the autoregressive draft during execution (Figure 2), therefore starts recovery as soon as feedback arrives and runs it in parallel with the check; an acceptance cancels it. Let C_V, C_A, and C_F be the costs of the check, of continuation after acceptance, and of recovery, excluding the history update that both branches share. With acceptance probability p,

$$\mathbb{E}[L] \approx p\,(C_V + C_A) + (1 - p)\,C_F. \tag{9}$$

Reuse therefore lowers mean latency below that of recovery alone whenever $C_V + C_A < C_F$, by an amount that grows with p. Returning the drafted action makes C_A negligible on video-conditioned models, and Section 5.3 measures both branches. Both branches, however, start only after the history update, while the main replica is otherwise idle during execution as the second one drafts. On models that keep a key/value history, the main replica therefore encodes the frames observed during execution as they arrive and appends them to a private copy of the history. Feedback commits this copy if it matches the reported observations and the published action, so only the last frames and the executed actions remain to be added (Figure 2; Appendix B).

5 Experiments

We evaluate the latency and task success of Spec-WAM on three WAMs (Section 5.2), then ablate feedback-grounded video verification and separate the sources of the latency reduction (Section 5.3).

5.1 Experimental setup

On RoboTwin 2.0 (Chen et al., 2025a), which has 50 tasks in clean and randomized settings, we evaluate 100 episodes per task drawn from the official evaluation seeds, with unseen instructions in the main comparison and seen instructions in the ablations. Configurations are compared on the same task–setting–seed units. LIBERO-Plus adds seven distribution shifts. Baselines are Fast-WAM (Yuan et al., 2026), Faster-WAM (Zhao et al., 2026), Motus (Bi et al., 2025), Cosmos-Policy on LIBERO-Plus (Kim et al., 2026), and the native inference of LingBot-VA, Next Forcing, and OpenWAM-α (Li et al., 2026; Xu et al., 2026; Wang et al., 2026), each on one GPU; Spec-WAM uses two (Section 4.4). On LingBot-VA and Next Forcing, drafts and recovery use four video steps and an accepted video returns its drafted action; on OpenWAM-α, drafts and recovery keep the released ten-step joint denoising schedule with velocity cache, and drafted actions are checked. Latency is the mean feedback-to-action latency (Section 3) on H100 GPUs without compilation; Appendix C lists the remaining settings.

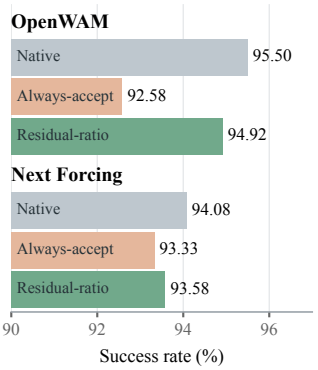

Figure 4: **Video-verification ablation.** Both speculative variants retain action checks; the residual-ratio gate uses $\tau_v = 0.2$. Runs use seen instructions.

5.2 Main quality–latency results

Across the three models of Table 2, Spec-WAM stays within 1 pp of the success of native inference, as it does on LIBERO-Plus (Table 3), because a draft is used only after it passes the check under the observations that arrive (Section 5.3). The latency reduction depends on how much sequential computation the native program performs after feedback. LingBot-VA and Next Forcing denoise video in 25 steps before inferring actions and take seconds per decision; four video steps with the batched action solver account for most of the reduction, and autoregressive

Table 3: **Success–latency comparison on LIBERO-Plus.** Success rate (%) under seven distribution shifts and mean latency (ms). †Success rates from the original publications; latencies are measured in our setup.

	Success rate (%) ↑								Lat. ↓
Method	Cam.	Robot	Lang.	Light	Bkg.	Noise	Lay.	Avg.	(ms)
Fast-WAM†	18.8	45.7	70.1	83.2	45.7	29.8	62.7	49.1	378.23
Faster-WAM†	53.8	71.6	94.7	96.3	61.3	63.6	79.1	73.6	298.33
Cosmos-Policy†	75.8	63.3	81.7	96.5	88.9	92.7	82.2	82.2	432.36
OpenWAM-α	36.0	77.5	88.2	98.0	87.9	41.3	78.5	70.4	284.70
+ Spec-WAM	37.1	78.1	88.2	97.5	89.0	42.6	77.8	70.8	220.40

Table 4: **Sources of the latency reduction on LingBot-VA.** Each row adds one change to native inference with 25 video steps; the last row is Spec-WAM. SR (%), ΔSR (pp) against native inference, and mean latency (ms).

	SR	ΔSR	mLat.
Native	90.25	–	6,383.76
+ 4 video steps, batched solver	89.83	−0.42	333.18
Spec-WAM	90.00	−0.25	176.14

drafts, prepared while the previous chunk executes, roughly halve the rest (Section 5.3). Next Forcing gains more from drafts because recovery keeps the drafted actions that pass the check under the regenerated video: after a rejection it keeps all 49 drafted transitions in 65.0% of cases, against 26.4% on LingBot-VA, so its rejected decisions take 261.13 ms instead of 320.06 ms. The released OpenWAM-α program already uses a velocity cache and responds in 286.13 ms. Spec-WAM keeps its ten-step joint schedule, so the reduction to 224.00 ms comes from drafts alone, and it is smaller because checking a draft costs a large share of the released program's latency.

Spec-WAM obtains these reductions **without additional training**, whereas Fast-WAM and Faster-WAM train models that generate less future video. With a second GPU, Next Forcing with Spec-WAM responds in 146.68 ms at 92.38% success, against 290.82 ms at 92.52% for Faster-WAM, while keeping future-video prediction. In Figure 1, the three Spec-WAM configurations are the fastest, and Next Forcing and OpenWAM-α with Spec-WAM lie on the Pareto frontier together with the released OpenWAM-α program.

5.3 Ablations

Effect of feedback-grounded video verification. Figure 4 compares native inference with two speculative variants: always accepting the drafted video and selecting it with the residual-ratio gate. Both speculative variants retain action checks, and all runs use seen instructions. On Next Forcing, native inference uses 25 video steps and the speculative variants use four. Video verification improves success over always-accept by 2.33 pp on OpenWAM and 0.25 pp on Next Forcing, reaching 94.92% and 93.58%, respectively. Native inference achieves 95.50% and 94.08%. Always accepting also reuses drafts whose predicted video diverges from execution, such as the rejected draft in Figure 3(c).

Sources of the latency reduction. Table 4 adds the parts of Spec-WAM one at a time on 1,200 of the unseen LingBot-VA episodes of Table 2. Four video steps with the batched action solver (Table 1) account for most of the reduction. Autoregressive drafts then lower mean latency by a further factor of 1.89, and success stays within 0.5 pp across the three rows. Accepted decisions take 91.89 ms, and rejected ones take 320.06 ms, close to the path without drafts (Appendix C). Next Forcing shows the same pattern (Table 5). On OpenWAM-α, shortening the released ten-step program to two or four steps instead lowers success to 89.67% and 90.17% (Appendix D).

6 Conclusion

Spec-WAM computes ahead along the denoising and execution axes and commits only after feedback: it checks analytic drafts in batched queries and autoregressive drafts, prepared during execution, under the new observation. Reuse follows each model's video–action coupling: an accepted video brings its drafted actions, and a joint model checks both streams in one query and resumes from the last verified state. Without additional training, Spec-WAM keeps success within 1 pp of native inference on three WAMs and lowers latency most where native inference computes the most after feedback.

AI USE STATEMENT

We used LLM-based assistants for experiment design, and drafting and editing text, figures, and code, but not for generating data or formulating the method and its claims. The authors reviewed all AI-assisted work, checked every reported number against the run logs, and take responsibility for the final content.

REPRODUCIBILITY STATEMENT

Spec-WAM requires no training and is applied to publicly released pretrained checkpoints of LingBot-VA, Next Forcing, and OpenWAM. Section 4 defines the drafting, verification, and scheduling rules, and Algorithm 1 in Appendix A gives the complete action solver with its constants. Appendix B specifies both video gates, and Appendix C lists every evaluated configuration, including denoising schedules, thresholds, compilation, and caching. Section 5.1 describes the RoboTwin 2.0 and LIBERO-Plus protocols, and Section 3 defines the latency boundary used for all timings.

REFERENCES

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, Hongyan Zhao, Hanyu Liu, Zhizhong Su, Lei Ma, Hang Su, and Jun Zhu. Motus: A unified latent action world model, 2025. URL https://arxiv.org/abs/2512.13030.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies, 2025. URL https://arxiv.org/abs/2506.07339.

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*, 2025a.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable, robust, and hardware-aware speculative decoding, 2025b. URL https://arxiv.org/abs/2402.12374.

Valentin De Bortoli, Alexandre Galashov, Arthur Gretton, and Arnaud Doucet. Accelerated diffusion models via speculative sampling. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 12590–12631. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/de-bortoli25a.html.

Yueci Deng, Guiliang Liu, and Kui Jia. Dexworldmodel: Causal latent world modeling towards automated learning of embodied tasks, 2026. URL https://arxiv.org/abs/2604.16484.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, and Jinwei Gu. 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, 2023. URL https://arxiv.org/abs/2211.17192.

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

Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model, 2025. URL https://arxiv.org/abs/2411.19108.

Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running vlas at real-time speed, 2025. URL https://arxiv.org/abs/2510.26742.

Jiahui Niu, Kefan Gu, Yucheng Zhao, Shengwen Liang, Tiancai Wang, Xing Hu, Ying Wang, and Huawei Li. Realtime-vla flash: Speculative inference framework for diffusion-based vlas, 2026. URL https://arxiv.org/abs/2605.13778.

Andy Shih, Suneel Belkhale, Stefano Ermon, Dorsa Sadigh, and Nima Anari. Parallel sampling of diffusion models, 2023. URL https://arxiv.org/abs/2305.16317.

Alexander Soen, Hisham Husain, Valentin De Bortoli, and Arnaud Doucet. Accelerating speculative diffusions via block verification, 2026. URL https://arxiv.org/abs/2606.13426.

Zhiwei Tang, Jiasheng Tang, Hao Luo, Fan Wang, and Tsung-Hui Chang. Accelerating parallel sampling of diffusion models. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 47800–47818. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/tang24f.html.

Rhoda AI Team. Causal video models are data-efficient robot policy learners. *Rhoda AI Blog*, 2026.

Songsheng Wang, Rucheng Yu, Zhihang Yuan, Chao Yu, Feng Gao, Yu Wang, and Derek F. Wong. Spec-vla: Speculative decoding for vision-language-action models with relaxed acceptance, 2025. URL https://arxiv.org/abs/2507.22424.

Yunsong Wang, Charlene Yang, Steven Farrell, Yan Zhang, Thorsten Kurth, and Samuel Williams. Time-based roofline for deep learning performance analysis. In *2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers (DLS)*, pp. 10–19, 2020. doi: 10.1109/DLS51937.2020.00007.

Yuran Wang, Siqiao Huang, Mingleyang Li, Chenhao Zhang, Jiaqi Liang, Weiyang Jin, Yue Chen, Xuemin Chi, Donghao Zhou, Qize Yu, Yu-Kai Wang, Yuhan Rui, Shenzhe Yao, Zhen Yuan, Zhenhao Shen, Kefei Zhu, Zijie Zhu, Ning Gao, Xiaowei Chi, Guanqi He, Shanghang Zhang, Hao Dong, Lin Shao, and Hang Zhao. Openwam: An open, modular exploration towards systematic world-action model pretraining. *arXiv preprint arXiv:2609.07398*, 2026.

Samuel Williams, Andrew Waterman, and David A. Patterson. Roofline: An insightful visual performance model for floating-point programs and multicore architectures. 2008. URL https://api.semanticscholar.org/CorpusID:5703612.

Gangwei Xu, Qihang Zhang, Jiaming Zhou, Xing Zhu, Yujun Shen, Xin Yang, and Yinghao Xu. Next forcing: Causal world modeling with multi-chunk prediction. *arXiv preprint arXiv:2606.11187*, 2026.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi "Jim" Fan, and Joel Jang. World action models are zero-shot policies, 2026. URL https://arxiv.org/abs/2602.15922.

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. URL https://arxiv.org/abs/2603.16666.

Qihang Zhang, Lin Li, Luyao Zhang, Shuai Yang, Yiming Luo, Shuaiting Li, Ruilin Wang, Junke Wang, Jiahao Shao, Gangwei Xu, Jiaming Zhou, Yishu Shen, Yudong Jin, Fangyi Xu, Shuailei Ma, Jiaqi Liao, Guanxing Lu, Zifan Shi, Yongkun Wen, Yujie Zhao, Weixuan Tang, Xinyang Wang, Chaojian Li, Jiapeng Zhu, Ka Leong Cheng, Nan Xue, Xing Zhu, Yujun Shen, and Yinghao Xu. Native video-action pretraining for generalizable robot control, 2026. URL https://arxiv.org/abs/2607.08639.

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*, 2026.

A Denoising Solver Details

Algorithm 1 specifies the fixed-window action solver used in Section 4.1. Its candidate transitions follow the original noise grid, and its analytic draft is given by Equation 2 with $\sigma_{\text{hold}} = 0.33$ and $\epsilon_{\text{num}} = 10^{-4}$. For LingBot-VA, the grid contains 50 transitions and a final cache-writing query; a verification window covers up to 49 transitions before the final target Euler step. We use the Euler operator

$$\mathrm{E}(x, v, \sigma \to \sigma') = x + (\sigma' - \sigma)v. \tag{10}$$

The algorithm writes ρ for the RMS over active normalized action coordinates.

Algorithm 1 Speculative action denoising with a defect budget

Input: Initial noise x_0; noise grid $\sigma_0 > \cdots > \sigma_S = 0$; target velocity v^*; drafter $\widehat{v}$; tolerance τ_a; window K
Output: Action chunk x_S

$j \leftarrow 0$; $x \leftarrow x_0$; $v \leftarrow v^*(x_0, \sigma_0)$; $D, L \leftarrow 0$
$\mathcal{H} \leftarrow [(\sigma_0, x, v)]$
while $j < S - 1$ **do**
  // Draft and evaluate
  $k \leftarrow \min(K, S - 1 - j)$
  $y_{0:k} \leftarrow \texttt{Draft}(x, \sigma_{j:j+k}, \mathcal{H}, \widehat{v})$
  $w_0 \leftarrow v$; $w_{1:k} \leftarrow v^*(y_{1:k}, \sigma_{j+1:j+k})$ ▷ One batched call
  // Verify the longest admissible prefix
  $\delta_r \leftarrow \rho\big(y_{r+1} - (y_r + h_{j+r} w_r)\big)$; $\ell_r \leftarrow \rho(h_{j+r} w_r)$ ▷ $0 \le r < k$
  $D_q \leftarrow D + \sum_{r=0}^{q-1} \delta_r$; $L_q \leftarrow L + \sum_{r=0}^{q-1} \ell_r$ ▷ $1 \le q \le k$
  $m \leftarrow \max\{r \in \{0, \ldots, k\} : D_q \le \tau_a L_q \text{ for all } 1 \le q \le r\}$
  // Reuse accepted progress or take a target step
  if $m > 0$ **then**
    Append $(\sigma_{j+r}, y_r, w_r)_{r=1}^{m}$ to $\mathcal{H}$
    $(x, v) \leftarrow (y_m, w_m)$ ▷ Reuse the evaluated velocity
    $(D, L, j) \leftarrow (D_m, L_m, j + m)$
  else
    $L \leftarrow L + \rho(h_j v)$; $x \leftarrow x + h_j v$
    $j \leftarrow j + 1$; $v \leftarrow v^*(x, \sigma_j)$ ▷ Refresh at the new state
    Append (σ_j, x, v) to $\mathcal{H}$
  end if
end while
$x_S \leftarrow x + h_{S-1} v$ ▷ Final Euler step to zero noise
return x_S

Backward-error certificate. The transition identity follows from the definition of e_i. Accepted prefixes satisfy Equation 8; fallback and the final target Euler transition add zero defect and nonnegative target Euler path length. Induction over committed transitions therefore preserves the inequality at every prefix.

Adaptive window extension. For $K = 8$, an optional heuristic extends the base window to at most $\min(32, S - 1 - j)$ candidates; verification still uses Equation 8. Let ϵ_i be the RMS difference between three-point and two-point velocity extrapolations from the fixed committed history. With two tuples, compare two-point extrapolation with the latest committed velocity; with one tuple, set $\epsilon_i = 0$. For each verified window w, record $P_w = \sum_i \epsilon_i$ and $R_w = \sum_i \|e_i\| / \max(\ell_i, 10^{-8})$. Across chunks, retain the last 32 pairs and set s to the upper median of $R_w / \max(P_w, 10^{-12})$. After eight recorded windows, append another candidate while $s \sum_{i<k} \epsilon_i \le \tau_a k/2$, where k is the current candidate count. This forecast selects batch size, not acceptance.

Local perturbation interpretation. For every committed transition, write $h_i = \sigma_{i+1} - \sigma_i$ and $\delta v_i = e_i/h_i$, assigning $e_i = 0$ to target Euler steps. The complete checked trajectory satisfies

$$\sum_{i<r} |h_i| \|\delta v_i\| \le \tau_a \sum_{i<r} |h_i| \|v_i^*\| \quad \text{at every committed prefix } r. \tag{11}$$

This bounds accumulated local perturbations along the committed path. A bound on separation from another trajectory additionally depends on the field's stability. Sparse cross-feedback routing evaluates selected positions; the full budget applies to the transitions checked by the solver after that routing decision.

Draft defect identity. At a drafted step with $\sigma_i \ge 10^{-4}$ and $\ell_i > 0$, let $\bar{x}_i = x_i - \sigma_i v_i^*$ and let $\widehat{\bar{x}}_i$ be the drafted clean estimate, held or extrapolated. Equation 3 then gives the relative local defect

$$\frac{\|e_i\|}{\ell_i} = \frac{\|\bar{x}_i - \widehat{\bar{x}}_i\|}{\|x_i - \bar{x}_i\|}. \tag{12}$$

The step-size factor cancels locally. Across a window, the acceptance rule weights these ratios by the target displacements ℓ_i and carries the incoming budget D, L.

Solver cost. With W verification windows of sizes k_w and F fallback refreshes,

$$\begin{aligned} T_{\text{action}} = T_{\text{init}} + T_{\text{commit}} + \sum_{w=1}^{W} T_{\text{verify}}(k_w) \\ + \sum_{f=1}^{F} T_{\text{refresh},f} + T_{\text{draft}} + T_{\text{accept}}. \end{aligned} \tag{13}$$

The terminal commit cost is present for models that require a cache-writing query. Verification cost includes preparing the batched inputs and processing the output fields. Measuring these terms distinguishes sequential call reduction from wall-clock speedup, particularly when video and action share a more expensive joint forward pass.

B Feedback Grounding and Video Gate

Observation and candidate state. The implementation maintains three contexts. `REAL` contains actual observations and executed actions; `CURRENT` is a private query and action-solve context; `NEXT` supplies the parent for speculative generation (Figure 5). Feedback first updates `REAL`. Verification and recovery queries then use a read-only view of this grounded context, keeping predicted video out of the observation history against which it is checked. Episode, chunk, parent-action, noise, and schedule identities associate each candidate with its intended feedback boundary. Incomplete or mismatched candidates are replaced by recovery.

Streaming history update. LingBot-VA and Next Forcing keep a key/value history of observed video and executed actions. Each history update encodes the new frames into video latents, adds their entries with a video forward, and adds the entries of the executed actions with an action forward, which attends to the complete video of the chunk. While u_t executes, the client sends each complete group of four camera frames as it arrives. The main replica encodes the group into one latent frame, runs the video forward on it, and appends the entries to a private copy of the `REAL` cache, which checks, drafts, and reanchoring cannot read. Frames of the last group are prepared on the host as they arrive and encoded together with the final frame. Feedback commits the staged entries only if the observations and the executed action they assume match the feedback; otherwise it discards them and runs the full update. The rest of the update encodes the last group, adds its video entries, and runs the action forward over the whole chunk. With two replicas, the second one runs this action forward layer by layer behind the main replica's video forward and produces the same entries. Staging is not numerically identical to the released update: the entries of a latent frame are computed before the next frame of the chunk arrives and do not attend to it, and separate encoder calls can differ in bfloat16. The Spec-WAM configurations of LingBot-VA and Next Forcing use this update. OpenWAM-α conditions each decision on the current observation and keeps no key/value history, so it has no staged update.

Action selection and reanchoring. A single accepted or recovered action is returned for execution. When recovery races verification, only the selected branch supplies the action. After accepted reuse, a background solve under `REAL` constructs a new parent from the actual observation and the selected action. This parent anchors the next autoregressive draft to the latest feedback. The candidate-ready condition includes completion of this preparation and any required state transfer.

Ratio-gate instance and selection. The ratio instance uses the candidate-update-normalized Euler residual in Equation 6, with the global region and the queried probe nearest the requested sigma 0.95. LingBot's four-step schedule maps the request to 0.9375; probe selection operates on saved grid positions. The value 0.2 was fixed as a shared tolerance after offline inspection of archived residuals and evaluation of candidate scoring rules, including a stricter 0.1 setting. The reported closed-loop comparison evaluates this combined first-probe/global instance, whose behavior is measured separately on each model.

For a selected region R and probe set $\mathcal{I}$, the more general implementation uses the ratio of summed RMS norms, $\sum_{i\in\mathcal{I}} \mathrm{RMS}_R(r_i^v) / \sum_{i\in\mathcal{I}} \mathrm{RMS}_R(d_i)$. Each informative region must pass. The ratio uses no denominator floor: a zero update with nonzero residual rejects, while a zero update and zero residual provide no score. If no informative region remains, recovery is selected. The ratio decision reads the selected scores from already computed queries; the fused video/action query union remains unchanged.

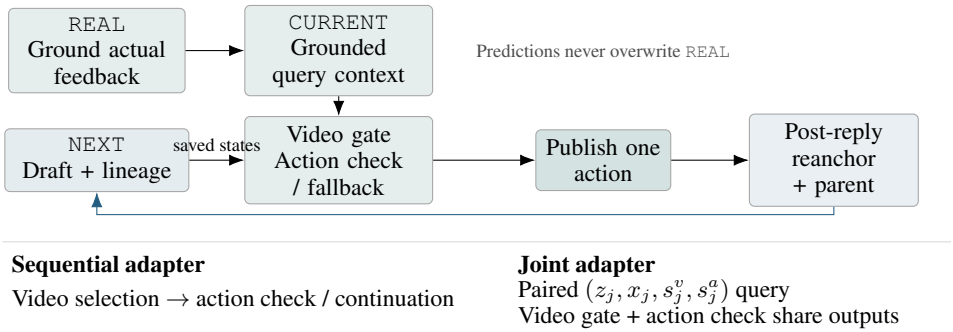

Figure 5: **Feedback-grounded execution.** Grounded history supplies checks and recovery, while speculative history prepares the next trajectory. Sequential and joint query paths preserve their model dependencies. Reanchoring constructs the next parent after the current action is selected.

C Evaluation Configurations

All configurations probe the video gate at $\sigma_{\mathrm{probe}} = 0.95$. The seen-instruction studies run on one H200 GPU without compilation, with $\tau_v = 0.2$ and checked actions unless stated otherwise. In the Table 2 configurations of LingBot-VA and Next Forcing, drafts were ready at most 787 ms after the preceding action was published.

LingBot-VA. The denoising-axis study uses one Next Forcing checkpoint, the same video-conditioning program for each action-solver comparison, and a 50-step action grid. The dense solver evaluates 50 transitions and one cache-writing query. Speculative variants use $\tau_a = 0.1$ and base windows $K = 8$ or $K = 49$; the $K = 8$ variant enables the adaptive extension described above. The two paired comparisons contain 1,122 and 1,124 scene-matched episodes. Full-protocol rates use all 1,200 episodes; matched-pair rates use only scenes available in both configurations.

The feedback-grounded configurations use the ratio gate, four video denoising steps for both recovery and draft generation, and video CFG 5. Their action grid contains 50 steps with action CFG 1. When actions are checked after feedback, the adapter compares saved and updated action velocities at a few saved states, together with a gripper check, and resumes from the selected state with a new defect budget; after video rejection, it can recheck the saved action states under the regenerated video. Five action probes at positions $0/12/24/36/49$ use a prefix threshold of 0.03, followed by defect-budget continuation with $\tau_a = 0.1$. The reference quality cohort uses the original 25-step

video and 50-step action program; Table 2 compares the complete inference configurations. The checked configuration performs action verification after the video gate accepts. In Table 2, LingBot-VA uses the ratio gate with $\tau_v = 0.25$ and Next Forcing with $\tau_v = 0.2$; both run on two replicas and return drafted actions after video acceptance.

Action reuse after video rejection. On the 30 timing units behind Table 2, accepted decisions take 92.32 ms on average for Next Forcing and 91.89 ms for LingBot-VA, whereas rejected decisions take 261.13 and 320.06 ms; rejections are similarly frequent (32.6% and 36.2% of steady decisions). The video regeneration and the gate cost the same on both checkpoints, and the difference lies in the action solve that follows. On a rejection, recovery evaluates the rejected draft's action trajectory at five saved states under the regenerated video, keeps the prefix of the 49 transitions whose accumulated defect stays below 0.03, and solves only the remaining steps. For Next Forcing, 65.0% of rejections keep all 49 transitions and none keeps zero; the mean kept prefix is 44.5 transitions, and recovery makes 0.35 solver calls per rejection. For LingBot-VA, 26.4% keep all 49 and 32% keep none; the mean kept prefix is 26.6 transitions, and recovery makes 1.00 solver call per rejection. Next Forcing's action velocities therefore change less when the conditioning video changes from the draft to the regenerated one. Its post-training on predicted context is consistent with this difference, but we did not test that cause.

OpenWAM-α. Recovery uses the released ten-step joint video–action schedule. The shared-query configurations use a nine-transition action window, batched video probes, and shared joint fields. The ratio-0.2 and always-accept-video instances both retain action checks and run on one replica without compilation or the native DiT velocity cache. The Table 2 configuration runs the ratio-0.2 instance on two replicas and keeps the DiT velocity cache in recovery and reanchoring. The task-success reference returns the released program's actions and records shadow checks; the latency reference runs the same program with those unused diagnostics disabled. The two-replica configuration performs verification on the main replica and recovery on the second replica.

Timing and execution windows. Steady-decision latency excludes the first decision, for which no preceding execution window can supply a candidate. Eager and compiled timings use separate native-inference references on the same host. Contention between the two replicas and the transfer of inputs to the second replica add to both branches of Equation 9. The nominal LingBot action horizon is 640 ms (32 slots at 50 Hz).

D Additional Results

Sources of the latency reduction on Next Forcing. Table 5 repeats the comparison of Table 4 on the 1,195 Next Forcing units of Table 2. Four video steps with the batched solver, computed after feedback, take 333.10 ms against 5,009.50 ms for native inference, at 91.88% against 91.63% success. Autoregressive drafts then lower mean latency to 146.68 ms on the same 30 timing units, a further factor of 2.27, at 92.38% success. Accepted decisions take 92.32 ms, and rejected ones take 261.13 ms, less than the 333.10 ms without autoregressive drafts because recovery keeps the saved action states that pass the check under the regenerated video (Appendix C).

Table 5: **Sources of the latency reduction on Next Forcing.** Rows as in Table 4, on the 1,195 unseen units of Table 2; ΔSR is against native inference. The last two rows are timed on the same 30 units and H100 host; native latency repeats Table 2.

	SR (%)	ΔSR (pp)	Mean latency (ms)
Native	91.63	–	5,009.50
+ 4 video steps, batched solver	91.88	+0.25	333.10
+ autoregressive drafts (Spec-WAM)	92.38	+0.75	146.68

Denoising-axis call reduction. The completed Next Forcing solver comparison retains the 50-step action grid and fixed video program (Table 6). Mean sequential target calls fall from 51 to 4.02 with $K = 49$, and 9,025 of 9,481 chunks use four calls. The paired success rates remain close to

the dense solver. These counts measure sequential query rounds; the batch still evaluates multiple states.

Table 6: **Denoising-axis call reduction on Next Forcing.** Calls include the final cache-writing query. Paired SR uses the matched episodes of each dense-versus-speculative comparison.

Solver	Mean sequential calls	Call reduction	Paired n	Paired SR (%)
Dense	51	$1.00\times$	–	–
Speculative, $K = 8$ + adaptive	6.02	$8.47\times$	1,122	$94.5 \rightarrow 94.9$
Speculative, $K = 49$	4.02	$12.69\times$	1,124	$94.7 \rightarrow 95.2$

Fewer native denoising steps on OpenWAM-α. Spec-WAM keeps the released ten-step joint program of OpenWAM-α, so shortening that program is the direct alternative to speculation. With the released server, compilation, and its velocity cache at four and two steps, success falls to 90.17% and 89.67% on 600 episodes.

Batch cost of model queries. Action-only and paired joint queries have different state sizes and conditioning costs. To test the batching condition $T_k < mT_1$ in Section 4.1, we time one production query over k states against the same query over one state, $\rho(k) = T_k/T_1$, on an otherwise idle H200. A LingBot-VA action state contributes 32 input rows per guidance lane, and ρ stays below 1.3 up to about 512 rows (16 states with one lane, 8 with two), beyond the nominal ridge of about 206 rows for weight reads. Past this point, attention over the history makes the cost grow with its length: a full 49-state window costs 2.71 and 3.58 single queries with two lanes for a short and a long history (one lane in Table 1). The LingBot-VA video gate evaluates 480 rows per state, so with a long history each additional probe adds about 20 ms, and one probe costs 1.24–1.95 times less than three.

Checked versus direct action reuse. Holding the video gate fixed isolates the action check. On OpenWAM-α with the calibrated gate and seen instructions, returning drafted actions directly succeeds in 94.75% of 1,200 episodes, versus 95.17% with checked continuation (-0.42 pp); both reuse drafts in 40.7% of steady decisions. The check shares the joint query that scores the video (Section 4.4). When actions are generated after video, checking instead requires installing the selected video and evaluating action states in separate queries, so the direct mode removes more work; the LingBot-VA and Next Forcing rows of Table 2 use it, and their success rates include its effect. To measure that work, the LingBot-VA Spec-WAM profile is run with checked actions and otherwise unchanged settings on the same 30 closed-loop timing units. The check raises the mean latency of an accepted decision from 91.89 to 169.87 ms and leaves rejected decisions unchanged (320.06 and 325.86 ms). Over all decisions, at 63.1% and 59.5% acceptance, the mean latency rises from 176.14 to 232.97 ms, still 1.4 times below the 333.18 ms without autoregressive drafts (Table 4). On Next Forcing with the calibrated gate, one replica, and seen instructions, direct actions succeed in 93.17% of 1,200 episodes against 95.00% with checked continuation (-1.83 pp). The direct mode thus trades the action check's latency against task success.

Per-task success rates. Tables 7 and 8 give the success rate of every RoboTwin 2.0 task for the configurations of Table 2.

Rollout examples. Figure 6 shows episodes of OpenWAM-α with Spec-WAM on RoboTwin 2.0. The four successful episodes pick, stack, and place objects; both failures occur on the mug-hanging task, where neither mug ends on the rack.

Table 7: **Per-task success rates (%) on RoboTwin 2.0, clean setting.** †From Zhao et al. (2026, Table A5). We run the other columns with unseen instructions, evaluating each model and its Spec-WAM configuration on the same task–seed units; Average is taken over these episodes, as in Table 2. The best result in each row is in bold.

Task	Faster-WAM†	Fast-WAM†	Motus†	LingBot-VA		Next Forcing		OpenWAM-α	
				Native	Spec-WAM	Native	Spec-WAM	Native	Spec-WAM
Adjust Bottle	**100**	**100**	89	98.3	98.3	**100**	**100**	**100**	**100**
Beat Block Hammer	97	99	95	98.3	95	**100**	91.7	**100**	**100**
Blocks Ranking RGB	**100**	**100**	99	98.3	**100**	**100**	**100**	**100**	**100**
Blocks Ranking Size	74	94	75	93.3	93.3	**100**	**100**	75	75
Click Alarmclock	99	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Click Bell	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Dump Bin Bigbin	97	97	95	95	86.7	**100**	83.3	**100**	**100**
Grab Roller	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Handover Block	93	95	86	**100**	98.3	**100**	**100**	91.7	91.7
Handover Mic	**100**	99	78	88.3	88.3	**100**	**100**	**100**	**100**
Hanging Mug	61	58	38	21.7	23.3	**91.7**	**91.7**	58.3	58.3
Lift Pot	**100**	**100**	96	**100**	**100**	**100**	**100**	**100**	**100**
Move Can Pot	96	90	34	88.3	98.3	75	83.3	**100**	**100**
Move Pillbottle Pad	98	**100**	93	**100**	98.3	**100**	91.7	**100**	**100**
Move Playingcard Away	**100**	**100**	**100**	**100**	96.7	**100**	**100**	**100**	**100**
Move Stapler Pad	82	77	83	70	66.7	83.3	83.3	**100**	**100**
Open Laptop	95	98	95	96.7	96.7	**100**	**100**	**100**	**100**
Open Microwave	77	62	95	60	30	66.7	91.7	**100**	83.3
Pick Diverse Bottles	90	80	90	90	85	91.7	**100**	75	75
Pick Dual Bottles	98	**100**	96	**100**	**100**	**100**	**100**	83.3	**100**
Place A2B Left	96	95	88	96.7	98.3	91.7	91.7	**100**	**100**
Place A2B Right	94	93	91	95	91.7	91.7	91.7	**100**	83.3
Place Bread Basket	89	91	91	91.7	98.3	**100**	**100**	**100**	**100**
Place Bread Skillet	93	90	86	95	96.7	91.7	83.3	**100**	91.7
Place Burger Fries	98	96	98	98.3	96.7	**100**	91.7	**100**	**100**
Place Can Basket	76	71	81	**91.7**	81.7	66.7	83.3	**91.7**	**91.7**
Place Cans Plasticbox	99	99	98	**100**	98.3	**100**	**100**	**100**	**100**
Place Container Plate	98	96	98	93.3	96.7	**100**	91.7	**100**	91.7
Place Dual Shoes	91	94	93	96.7	91.7	91.7	91.7	91.7	**100**
Place Empty Cup	**100**	**100**	99	**100**	**100**	**100**	**100**	**100**	**100**
Place Fan	98	96	91	96.7	91.7	**100**	91.7	**100**	**100**
Place Mouse Pad	86	83	66	93.3	96.7	**100**	**100**	**100**	**100**
Place Object Basket	80	89	81	90	85	83.3	**100**	91.7	83.3
Place Object Scale	92	90	88	96.7	**100**	83.3	91.7	**100**	**100**
Place Object Stand	95	90	98	96.7	**100**	**100**	**100**	91.7	**100**
Place Phone Stand	**100**	97	87	98.3	98.3	**100**	**100**	91.7	83.3
Place Shoe	97	96	99	96.7	**100**	**100**	**100**	**100**	**100**
Press Stapler	90	90	93	80	83.3	75	66.7	**100**	**100**
Put Bottles Dustbin	94	95	81	76.7	86.7	83.3	66.7	91.7	**100**
Put Object Cabinet	87	**94**	88	78.3	80	75	91.7	83.3	91.7
Rotate QRcode	94	93	89	93.3	**96.7**	91.7	91.7	75	91.7
Scan Object	91	89	67	**95**	90	91.7	91.7	83.3	75
Shake Bottle	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Shake Bottle Horizontally	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Stack Blocks Three	98	95	91	98.3	**100**	**100**	**100**	91.7	**100**
Stack Blocks Two	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Stack Bowls Three	83	80	79	86.7	86.7	75	83.3	**91.7**	66.7
Stack Bowls Two	98	92	98	96.7	93.3	**100**	91.7	**100**	91.7
Stamp Seal	90	90	93	98.3	96.7	**100**	**100**	91.7	91.7
Turn Switch	75	61	**84**	53.3	43.3	75	58.3	83.3	75
Average	92.78	91.88	88.66	91.63	90.67	93.50	93.33	**94.66**	93.82

Table 8: **Per-task success rates (%) on RoboTwin 2.0, randomized setting.** Columns as in Table 7.

Task	Faster-WAM†	Fast-WAM†	Motus†	LingBot-VA		Next Forcing		OpenWAM-α	
				Native	Spec-WAM	Native	Spec-WAM	Native	Spec-WAM
Adjust Bottle	99	**100**	93	96.6	98.3	**100**	**100**	**100**	**100**
Beat Block Hammer	98	97	88	96.3	**100**	**100**	**100**	**100**	**100**
Blocks Ranking RGB	**100**	**100**	97	94.6	96.4	**100**	**100**	**100**	**100**
Blocks Ranking Size	93	**98**	63	87.7	87.7	91.7	91.7	75	75
Click Alarmclock	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Click Bell	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Dump Bin Bigbin	98	96	91	91.4	89.7	91.7	**100**	91.7	**100**
Grab Roller	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Handover Block	**88**	81	73	87.3	83.6	58.3	83.3	83.3	83.3
Handover Mic	99	**100**	63	94.7	96.5	**100**	**100**	91.7	91.7
Hanging Mug	44	62	38	37.9	29.3	**75**	66.7	58.3	50
Lift Pot	**100**	**100**	99	**100**	**100**	**100**	**100**	**100**	**100**
Move Can Pot	98	88	74	89.7	91.4	**100**	91.7	**100**	**100**
Move Pillbottle Pad	99	99	96	96.6	98.3	**100**	**100**	**100**	**100**
Move Playingcard Away	**100**	**100**	96	**100**	98.2	**100**	**100**	**100**	**100**
Move Stapler Pad	74	64	85	75.9	69.0	41.7	75	**91.7**	**91.7**
Open Laptop	99	**100**	91	96.5	96.5	**100**	**100**	**100**	**100**
Open Microwave	77	45	**91**	82.1	85.7	83.3	75	83.3	58.3
Pick Diverse Bottles	89	85	**91**	86.2	84.5	75	66.7	66.7	83.3
Pick Dual Bottles	98	96	90	**100**	93.2	91.7	**100**	91.7	**100**
Place A2B Left	94	93	79	91.1	92.9	83.3	75	**100**	**100**
Place A2B Right	93	99	87	89.7	91.4	91.7	91.7	**100**	**100**
Place Bread Basket	96	93	94	96.5	94.7	91.7	**100**	91.7	83.3
Place Bread Skillet	87	93	83	89.3	92.9	91.7	**100**	83.3	91.7
Place Burger Fries	97	99	98	94.7	96.5	**100**	**100**	91.7	91.7
Place Can Basket	73	69	76	88.7	86.8	75	**91.7**	66.7	66.7
Place Cans Plasticbox	94	96	94	**100**	98.3	**100**	**100**	91.7	91.7
Place Container Plate	97	**100**	99	94.8	98.3	**100**	**100**	**100**	**100**
Place Dual Shoes	88	88	87	88.9	83.3	**100**	91.7	91.7	91.7
Place Empty Cup	**100**	**100**	98	**100**	**100**	**100**	**100**	**100**	**100**
Place Fan	93	**96**	87	90.9	89.1	83.3	83.3	83.3	83.3
Place Mouse Pad	94	89	68	96.5	89.5	**100**	91.7	**100**	**100**
Place Object Basket	80	88	87	89.1	81.8	75	83.3	83.3	**91.7**
Place Object Scale	97	97	85	91.2	**100**	**100**	**100**	**100**	91.7
Place Object Stand	96	94	97	94.5	90.9	**100**	**100**	**100**	90.9
Place Phone Stand	96	99	86	96.6	94.8	**100**	91.7	91.7	91.7
Place Shoe	98	99	97	**100**	96.4	**100**	**100**	**100**	**100**
Press Stapler	92	97	**98**	84.2	84.2	58.3	41.7	91.7	91.7
Put Bottles Dustbin	91	90	79	86.7	88.3	91.7	91.7	**100**	91.7
Put Object Cabinet	89	89	71	76.3	76.3	58.3	**91.7**	58.3	**91.7**
Rotate QRcode	87	89	73	86.0	80.7	83.3	**91.7**	**91.7**	83.3
Scan Object	90	92	66	84.3	86.3	91.7	91.7	83.3	**100**
Shake Bottle	**100**	**100**	97	**100**	**100**	**100**	**100**	91.7	**100**
Shake Bottle Horizontally	**100**	**100**	98	**100**	98.0	**100**	**100**	**100**	91.7
Stack Blocks Three	96	97	95	96.4	96.4	**100**	**100**	83.3	91.7
Stack Blocks Two	**100**	**100**	98	**100**	**100**	**100**	**100**	**100**	**100**
Stack Bowls Three	78	81	**87**	75.9	86.2	75	75	83.3	66.7
Stack Bowls Two	96	98	98	98.3	98.3	**100**	**100**	**100**	91.7
Stamp Seal	92	94	92	94.6	98.2	**100**	**100**	91.7	**100**
Turn Switch	76	59	**78**	51.7	39.7	33.3	41.7	50	58.3
Average	**92.26**	91.78	87.02	90.71	90.07	89.75	91.43	90.65	91.15

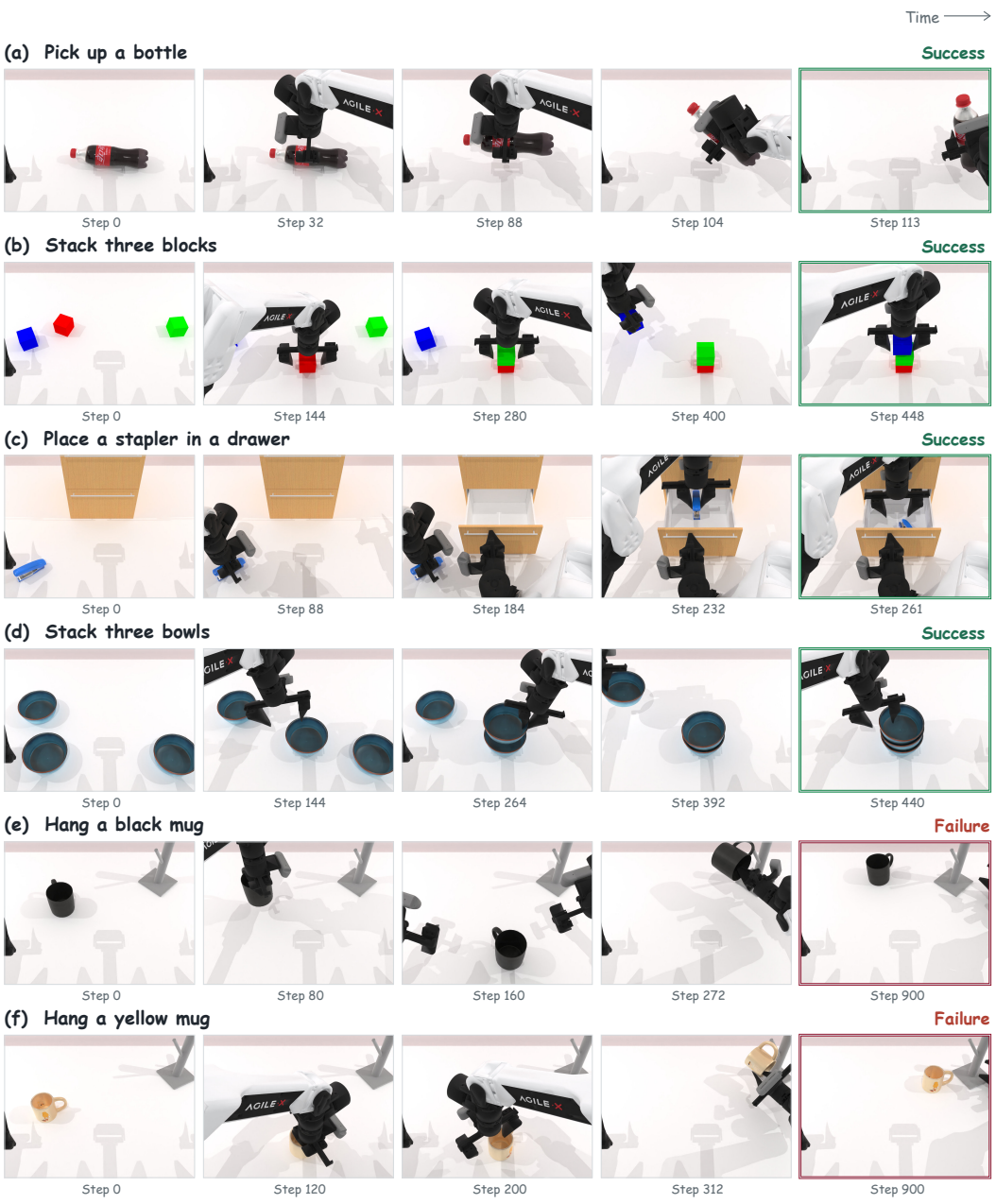

Figure 6: **OpenWAM-α with Spec-WAM on RoboTwin 2.0.** Four successful episodes (a–d) and two failed mug-hanging episodes (e, f), from the first to the final frame of each episode.