AHA-WAM: ASYNCHRONOUS HORIZON-ADAPTIVE WORLD-ACTION MODELING WITH OBSERVATION-GUIDED CONTEXT ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

World-action models have emerged as a promising paradigm for robot manipulation, jointly modeling visual scene dynamics and actions to inject physical priors into policy learning. However, existing world-action models couple world prediction and action execution at the same temporal resolution, forcing the world branch to model near-term frame variations that are redundant and weakly informative. We posit that strictly binding world prediction and action execution to the same temporal rhythm may underutilize the potential of the video branch for embodied control. Therefore, we propose AHA-WAM, an **A**synchronous **H**orizon-**A**daptive World-Action Model built on a dual Diffusion Transformer (DiT) architecture that reorganizes world-action modeling around this temporal asymmetry. AHA-WAM instantiates the video DiT as a low-frequency world planner that maintains rolling key-value memory over past observations and exposes reusable layerwise latent context encoding long-horizon scene evolution, while a high-frequency action DiT executes short action chunks in closed loop by querying this context through layerwise joint attention. To support asynchronous execution, we introduce horizon-adaptive offset training and Observation-Guided Video-Context Routing (OVCR), which together let the action expert exploit long-horizon world context while remaining responsive to real-time execution state without rerunning the video DiT. Experiments on RoboTwin and real-world manipulation tasks show that AHA-WAM achieves state-of-the-art performance without any robot-data pretraining, attaining **92.80%** average success on RoboTwin and **78.3%** success across 4 real-world tasks, while reaching **24.17** Hz closed-loop control with a **4.59**$\times$ speedup over Fast-WAM.

1 INTRODUCTION

Robotic manipulation requires policies that understand not only the current scene, but also how the scene may evolve under the robot's actions. While VLA models benefit from large-scale imitation learning, they receive limited supervision on physical dynamics with only action labels, while world-action models (WAMs) address this by coupling action prediction with dense video-based world modeling, learning how actions co-evolve with visual scene dynamics to inject physical priors into control and offer a more generalizable, and transferable policy representation.

Despite this promise, current WAMs leave an important design space underexplored: how can the video branch more effectively empower robot control? Existing approaches either explicitly roll out future frames for inverse-dynamics decoding or joint world-action modeling, while others reuse the video branch as a latent encoder alongside the action branch. Both designs assume that world prediction and action execution share the same short horizon, which forces the world branch to spend capacity on dense adjacent-frame variations that are often highly correlated and only weakly informative for control. We argue that this coupling reflects a structural mismatch in temporal abstraction: the video world model can better serve embodied control by forming a temporally extended latent plan over future visual states, whereas the action model should remain tightly coupled to the real-time control loop, incorporating the latest observation to issue timely closed-loop corrections.

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.

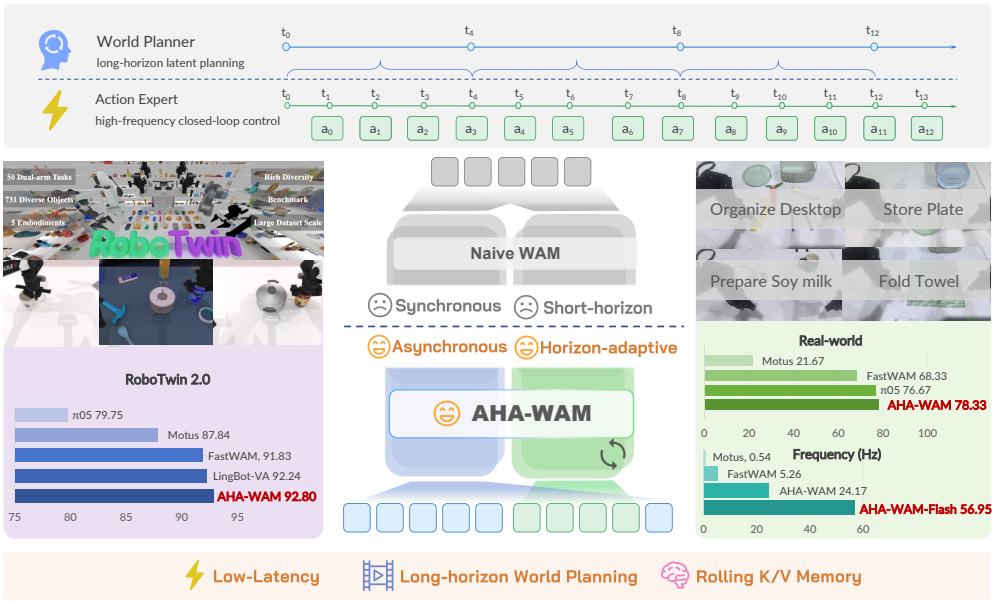

Figure 1: **Overview of AHA-WAM.** AHA-WAM connects past observations, future-oriented world planning, and fast closed-loop action execution: a slow world planner maintains reusable memory and planning context, while a fast action expert adapts that context to the latest observation before predicting short action chunks.

We propose AHA-WAM, an **A**synchronous **H**orizon-**A**daptive World-Action Model that reorganizes world-action modeling around this temporal asymmetry. Rather than just a scheduling strategy, AHA-WAM introduces an asynchronous modeling interface between predictive world representation and reactive action generation. AHA-WAM instantiates the video Diffusion Transformer (DiT) as a low-frequency world planner that maintains a rolling key-value (KV) memory over past observations and exposes reusable layerwise latent context encoding long-horizon scene evolution, amortizing expensive world-model computation across multiple action steps. In parallel, a high-frequency action DiT executes short action chunks in closed loop by querying this context through layerwise joint attention. This design preserves the benefits of joint world-action modeling while aligning each branch with the temporal scale at which it's most informative. In AHA-WAM, *horizon-adaptive* refers not to online horizon adjustment, but to a horizon-decoupled formulation where the world and action branches are assigned different temporal horizons according to their functional roles.

A key challenge introduced by asynchronous execution is that the planner context may become stale or misaligned with the current action chunk as the executor runs ahead. To address this, we introduce Observation-Guided Video-Context Routing (OVCR), which constructs chunk-specific latent queries from the latest visual observations to route and update the planner's layerwise video context before it is consumed by the action DiT, giving each action prediction an observation-conditioned view of the slow planner state without feeding dense visual tokens into the high-frequency action branch. Proprioceptive feedback enters the action DiT directly, while visual feedback is injected indirectly through the routed video context, keeping the reused planner context responsive to real-time execution state without recomputation. We further introduce horizon-adaptive offset training to expose the model to diverse planner-executor phase relationships, and equip the video DiT with rolling KV memory to connect past observations with future plans.

We evaluate AHA-WAM on RoboTwin 2.0 and real-world manipulation tasks. Though without robot-data pretraining, AHA-WAM achieves 92.80% average success across 50 RoboTwin 2.0 tasks, reaching state-of-the-art-level performance among strong VLA and WAM baselines. Across four real-world tasks covering deformable manipulation, long-horizon organization, fine-grained tool use, and spatial generalization, AHA-WAM achieves 78.3% average success, demonstrating robust deployment performance beyond simulation. Across four dimensions of real-world out-of-distribution evaluation, AHA-WAM matches $\pi_{0.5}$ in exhibiting the most limited performance degradation, suggesting stronger robustness to distribution shifts. Benefiting from the asynchronous infer-

ence schedule, ODE distillation and cuda optimizations, AHA-WAM reaches up to 56.9 Hz closed-loop control frequency, corresponding to a 10.82× speedup over Fast-WAM.

In summary, our contributions are threefold:

- We identify and address a fundamental temporal mismatch in existing world-action models, where world prediction and action execution are constrained to share the same temporal horizon. We introduce AHA-WAM, an asynchronous horizon-adaptive formulation that decouples long-horizon world planning from high-frequency action execution.
- We develop Observation-Guided Video-Context Routing (OVCR), which dynamically constructs chunk-specific latent video context from current observations to keep asynchronous planner context aligned with real-time execution state with horizon-adaptive offset training.
- We demonstrate that asynchronous world-action modeling enables both strong performance and efficient deployment. AHA-WAM achieves competitive results on simulation benchmarks and real-world robot tasks while substantially reducing latency through streaming inference.

2 Related Work

Generalist robot manipulation policies. Recent vision-language-action (VLA) policies leverage large-scale demonstrations and pretrained vision-language models to improve generalization across manipulation tasks (Brohan et al., 2023; Black et al., 2024; Kim et al., 2025; Physical Intelligence et al., 2025). Despite their scalability, these methods remain largely action-centric, with physical dynamics learned only implicitly from action supervision. World-action models extend this paradigm by jointly modeling visual dynamics and actions, introducing explicit dense priors for robot control.

World models for robot control. Existing robot world models mainly follow two paradigms: *imagine-then-act*, which predicts future visual states or representations for downstream action generation (Du et al., 2023; Tian et al., 2025; Hu et al., 2025), and *joint world-action modeling*, which learns visual evolution and action generation in a unified architecture (Ye et al., 2026b; Bi et al., 2025; Kim et al., 2026; Li et al., 2026; Yuan et al., 2026; Ye et al., 2026a). However, existing WAMs typically couple world prediction and action execution at the same temporal horizon, limiting the video branch to short-term dynamics and constraining its role in control. AHA-WAM addresses this limitation by decoupling long-horizon world planning from high-frequency action execution.

Dual-system and asynchronous robot policies. Recent works explore slow-fast or asynchronous designs for efficient robot control. RoboDual (Bu et al., 2024), Reactive Diffusion Policy (Xue et al., 2025), and AsyncVLA (Hirose et al., 2026) introduce slow components for guidance, representation, or feedback alongside fast controllers. Different from these approaches, AHA-WAM establishes asynchrony within joint world-action modeling: the video DiT serves as a reusable latent world planner, while the action DiT performs high-frequency closed-loop execution through planner-context interaction.

3 Asynchronous Horizon-Adaptive World-Action Modeling

World-action models couple action prediction with learned visual dynamics, but existing formulations typically organize the video branch and the action branch under the same short-horizon execution rhythm. We instead formulate world-action modeling as a two-timescale generation problem. Figure 2 illustrates the overall architecture. AHA-WAM reorganizes WAM inference into an asynchronous world-action coupling framework while preserving a dual-DiT architecture: a low-frequency video planner produces reusable long-horizon planner context, and a high-frequency action expert consumes this context for closed-loop action denoising. Here, "horizon-adaptive" denotes robustness to arbitrary planner–executor phase offsets under decoupled temporal horizons, rather than online horizon selection. In other words, AHA-WAM does not dynamically choose the video or action horizon at test time; instead, it trains the executor to consume long-horizon planner context under variable action-start offsets, making the asynchronous interface robust to the phase misalignment induced by streaming deployment.

This design is supported by three complementary mechanisms. Observation-Guided Video-Context Routing (OVCR) adapts the cached planner context to the latest observation without rerunning the

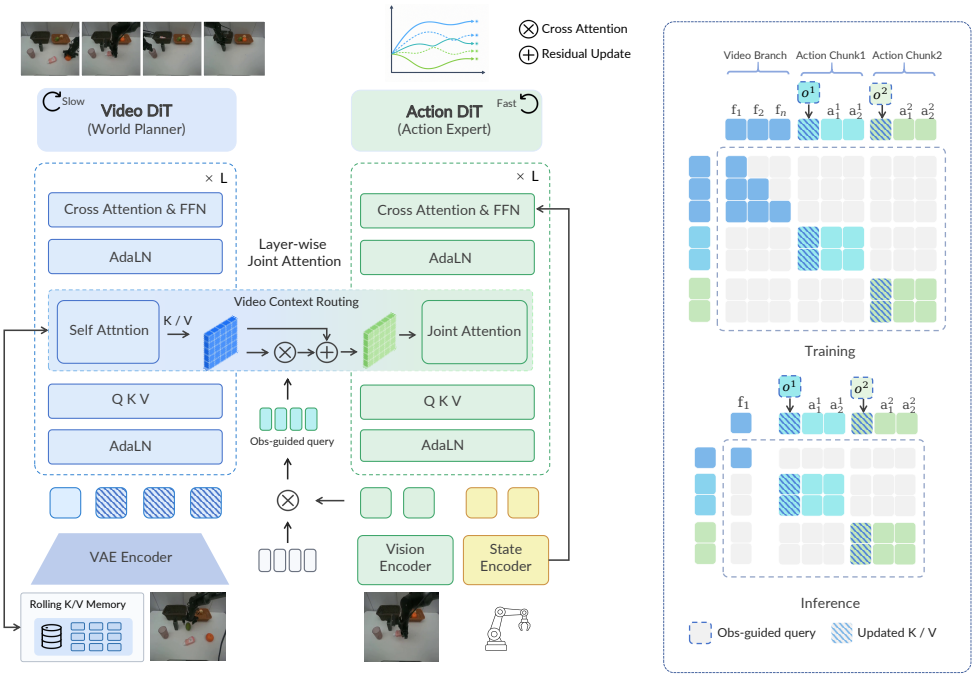

Figure 2: **AHA-WAM architecture and attention mask.** AHA-WAM decouples world planning and action execution into a slow video-DiT planner and a fast action-DiT executor. The video branch is trained with a fully causal mask to learn temporal dynamics. For each action update, the latest observation o^n first queries and updates the video DiT's K/V states through OVCR, producing updated planner context that is consumed by the action DiT through layerwise joint attention.

video DiT. Horizon-adaptive offset training exposes the action expert to the planner–executor phase shifts induced by asynchronous streaming, while rolling K/V memory extends the video planner's temporal receptive field over past observations. Finally, real-time inference optimizations further accelerate the high-frequency action stream for closed-loop deployment.

3.1 Dual-DiT Planner–Executor Architecture

Model architecture. AHA-WAM is built on a dual-DiT architecture (Peebles & Xie, 2023) consisting of a video DiT world planner and an action DiT executor. Visual observations are encoded by the pretrained VAE, while language embeddings from the text encoder condition both branches. The action branch is a lightweight action expert DiT with the same layer depth as the video DiT, enabling layerwise interaction between the two branches.

The video DiT takes visual tokens as input and is trained to predict future video latents over the longer planning horizon. The action DiT receives noisy action tokens and proprioceptive tokens, and denoises the action chunk under closed-loop feedback. Visual feedback for the high-frequency action branch is not introduced by naively concatenating dense visual tokens; instead, visual information is mediated through the planner video context and later adapted by OVCR in Section 3.2.

Layerwise planner–executor coupling. The central interface between the two DiTs is the layerwise planner video context. Given the visual observation context and language instruction, the video planner produces

$$\mathcal{C}_\tau^p = \left\{ \left(K_\tau^{p,\ell}, V_\tau^{p,\ell} \right) \right\}_{\ell=1}^{L}, \tag{1}$$

where p denotes the planner branch, τ identifies a planner refresh, and ℓ indexes one of L transformer layers. This context is a latent world-plan representation exposed by one video-DiT forward and reused by multiple subsequent action-DiT forwards. It differs from the rolling K/V memory in Section 3.4, which stores historical video states across planner refreshes inside the video planner.

During training, the video branch predicts future video latents under a fully causal video mask, encouraging the video DiT to learn forward scene dynamics while shaping the planner context with visual dynamics supervision. The action branch is masked from attending to future video tokens so that during inference, the future-video prediction path can be removed. Thus, future-video prediction serves as a world-modeling training signal, while planner video context serves as the inference-time interface to the action executor.

At each action update, the raw planner context $\mathcal{C}_\tau^p$ is first adapted by OVCR into a chunk-specific context $\widetilde{\mathcal{C}}_t^p = \{(\widetilde{K}_t^{p,\ell}, \widetilde{V}_t^{p,\ell})\}_{\ell=1}^L$. The action DiT then denoises the current action chunk through layerwise joint attention:

$$\bar{H}_t^{a,\ell} = \mathrm{Attn}\left(Q_t^{a,\ell}, \left[K_t^{a,\ell}; \widetilde{K}_t^{p,\ell}\right], \left[V_t^{a,\ell}; \widetilde{V}_t^{p,\ell}\right]\right), \tag{2}$$

where $Q_t^{a,\ell}$, $K_t^{a,\ell}$, and $V_t^{a,\ell}$ are the action-DiT query, key, and value projections, and $\bar{H}_t^{a,\ell}$ is the planner-conditioned action hidden state. This coupling preserves WAM-style interaction between visual dynamics and action generation, while amortizing expensive video-DiT computation across multiple high-frequency action updates.

Training objective. AHA-WAM jointly trains the action executor and video planner with a flow-matching objective over action chunks and future video latents. Specifically, the objective is defined as

$$\mathcal{L} = \mathcal{L}_{\mathrm{FM}}(A_t) + \lambda \mathcal{L}_{\mathrm{FM}}(Z_{t:t+h_v}^v), \qquad \lambda = 1, \tag{3}$$

where $\mathcal{L}_{\mathrm{FM}}(\cdot)$ denotes the flow-matching loss, A_t is the target action chunk, and $Z_{t:t+h_v}^v$ represents future video latents over the planning horizon h_v. During deployment, future-video prediction serves only as training supervision; the video DiT refreshes latent planner context, which is reused by the action DiT for closed-loop action generation without explicit future-frame decoding.

3.2 Observation-Guided Video-Context Routing

Asynchronous execution reuses one planner video context for multiple action chunks, which amortizes video-DiT computation but creates a context-alignment problem: before the next planner refresh, the robot state and visual scene may have changed. Observation-Guided Video-Context Routing (OVCR) addresses this by using the latest visual observation to convert the shared planner context into a chunk-specific context. Instead of feeding dense visual tokens into the high-frequency action DiT, OVCR compresses them into a small set of routing queries that select and edit the relevant planner context before action denoising.

For each action chunk at time t, we split the aligned observation context into visual tokens X_t^v and proprioceptive tokens X_t^s. The proprioceptive input is compact and directly tied to the instantaneous robot state, so a lightweight encoder maps it to a state token which is directly provided to the action DiT. Visual feedback is injected indirectly through context routing. Given Q learnable base queries $B \in \mathbb{R}^{Q\times d}$, OVCR constructs observation-guided routing queries by attention pooling over the current visual tokens:

$$Z_t^q = \mathrm{Attn}\left(B, f_v(X_t^v), f_v(X_t^v)\right), \tag{4}$$

where f_v projects visual tokens to the routing width d, and Q is the number of routing queries. The resulting queries provide compact, observation-conditioned slots for retrieving chunk-relevant information from the planner video context.

Let $\mathcal{C}_{\tau(t)}^p = \{(K_{\tau(t)}^{p,\ell}, V_{\tau(t)}^{p,\ell})\}_{\ell=1}^L$ denote the latest available planner context, with $\tau(t)$ identifying the most recent refresh available at action-update time t. For each layer ℓ, OVCR first reads planner features using the routing queries and then predicts residual key–value updates:

$$R_t^\ell = \mathrm{Attn}\left(Z_t^{q,\ell}, K_{\tau(t)}^{p,\ell}, V_{\tau(t)}^{p,\ell}\right), \qquad (\Delta K_t^{p,\ell}, \Delta V_t^{p,\ell}) = g_\psi^\ell\left(R_t^\ell, Z_t^{q,\ell}\right), \tag{5}$$

where g_ψ^ℓ is a lightweight layerwise router. The chunk-specific planner context is produced by a gated residual update:

$$\widetilde{K}_t^{p,\ell} = K_{\tau(t)}^{p,\ell} + \alpha_t^\ell \Delta K_t^{p,\ell}, \qquad \widetilde{V}_t^{p,\ell} = V_{\tau(t)}^{p,\ell} + \alpha_t^\ell \Delta V_t^{p,\ell}, \tag{6}$$

where α_t^ℓ is a learned gate. The adapted context $\widetilde{\mathcal{C}}_t^p = \{(\widetilde{K}_t^{p,\ell}, \widetilde{V}_t^{p,\ell})\}_{\ell=1}^L$ is then consumed by the action DiT through the layerwise joint attention in Eq. 2.

OVCR turns planner-context reuse from static caching into observation-conditioned retrieval and adaptation. Thus, the video DiT remains outside the per-update critical path, while each action chunk still receives a planner representation aligned with the latest visual evidence.

3.3 Horizon-Adaptive Offset Training

Asynchronous streaming changes the relative temporal phase between the slow planner and the fast executor. If training always uses a fixed alignment between the video planning window and the action chunk, the action DiT may overfit to a single planner–executor phase and become brittle when reusing planner context at intermediate phases during deployment. We therefore introduce horizon-adaptive offset training to expose the executor to the phase shifts induced by asynchronous inference.

Let h_v denote the video planning horizon and h_a denote the action chunk horizon, with $h_a < h_v$. For each training segment starting at time τ, the video planner models future video latents over $[\tau, \tau + h_v)$. Instead of aligning the first action chunk to the planner start, we sample

$$\delta \sim \mathcal{U}\{0, 1, \ldots, h_a - 1\}, \tag{7}$$

and shift the action-chunk grid by δ within the planner horizon. The action objective is then evaluated over offset-aligned chunks:

$$\mathcal{L}_a = \mathbb{E}_\delta \left[\mathcal{L}_{\mathrm{FM}} \left(A_\tau^\delta ; c_{\tau,\delta}^a \right) \right], \tag{8}$$

where A_τ^δ is an offset-aligned action chunk and $c_{\tau,\delta}^a$ contains its proprioception, language instruction, and observation-adapted planner context.

Sampling δ varies the action-start phase within one chunk period; it does not by itself cover arbitrary planner delays. OVCR separately adapts the reused context to the observation at each action update.

Since the planner–executor alignment is periodic with the action chunk size, sampling $\delta \in [0, h_a)$ covers all chunk-level phases encountered when one planner context is reused across multiple executor updates. Thus, horizon-adaptive offset training teaches the action DiT to consume long-horizon planner context under variable action-start offsets, while OVCR handles observation-conditioned context selection at each phase.

3.4 Rolling Planner Memory

The video planner maintains a FIFO K/V memory over recent planner states to preserve historical scene context across low-frequency refreshes. At each refresh, the planner attends to stored states when producing new planner context, then updates the memory by appending the latest states and discarding the oldest entries when the capacity is exceeded.

The memory is internal to the video DiT and is not directly accessed by the action executor, which only consumes the latest planner context after OVCR adaptation. Therefore, rolling memory extends the temporal receptive field of the slow planner, while OVCR provides observation-conditioned context alignment for the fast executor.

3.5 Streaming Inference and Real-Time Optimization

The asynchronous planner–executor design enables a streaming inference schedule, where video-DiT context refreshes run asynchronously while the action executor maintains the closed-loop control frequency. We measure the end-to-end latency of one action update, L_{chunk}, including image encoding, planner-context access, OVCR routing, and action denoising, with video-DiT prefill executed in the background.

CUDA deployment optimizations compile the repeated action inference path into an efficient execution pipeline, reducing the 10-step action latency from 415.77 ms to 41.37 ms. Building on this accelerated path, AHA-WAM-Flash distills the action sampler into two denoising steps while preserving the same OVCR interface, further reducing L_{chunk} to 17.56 ms. Detailed optimization breakdowns, measurement protocols, and distillation settings are provided in Appendix D.

4 EXPERIMENTS

We evaluate AHA-WAM in simulation and on real robots to test whether asynchronous world-action modeling preserves manipulation performance while improving closed-loop efficiency. We cover RoboTwin 2.0 (Chen et al., 2025) performance, component ablations, real-world deployment, generalization analysis and inference latency.

Specifically, our experiments ask five questions including capability, mechanism, deployability, robustness, and efficiency. **Q1.** Can our asynchronous WAM preserve or improve simulation performance while faster? (Sec. 4.2) **Q2.** How important is each component for AHA-WAM? (Sec. 4.3) **Q3.** Can AHA-WAM achieve reliable real-robot performance? (Sec. 4.4) **Q4.** How robust is AHA-WAM under previously unseen real-world task conditions? (Sec. 4.4) **Q5.** How much closed-loop latency reduction does AHA-WAM provide? (Sec. 4.5)

4.1 EXPERIMENTAL SETUP

Model and training. AHA-WAM is implemented as a dual-DiT policy with an explicitly asynchronous planner–executor interface. We use the pretrained Wan2.2-5B video model to initialize the world-planning branch, including the video DiT, text encoder, and video VAE. Following Fast-WAM (Yuan et al., 2026), the high-frequency action executor adopts a compact DiT architecture with hidden dimension $d_a = 1024$, corresponding to approximately 1.02B parameters. In addition to the 4.99B-parameter video planner and the 1.02B-parameter action DiT, AHA-WAM introduces 1.22B additional parameters for context routing and interface modules, resulting in a total model size of approximately 7.23B parameters.

The video branch operates over a long planning horizon of $h_v = 64$, while the action branch predicts short executable chunks with horizon $h_a = 16$. The action executor adapts this context through OVCR, using 32 observation-guided routing queries for each action chunk.

Latency is reported on a single NVIDIA RTX 5090D GPU. Appendix A lists additional implementation details, including AHA-WAM-Flash distillation and other training configurations.

Evaluation scope. We evaluate AHA-WAM in both RoboTwin 2.0 simulation (Chen et al., 2025) and real-world experiment. The simulation benchmark tests multi-task manipulation under clean and randomized scenes, while the real-world experiments test deployment on physical bimanual tasks. Across both settings, we report task success rate as the primary metric and additionally analyze closed-loop inference latency to quantify deployment efficiency.

4.2 ROBOTWIN SIMULATION

We evaluate AHA-WAM on RoboTwin 2.0 (Chen et al., 2025), a bimanual manipulation benchmark with 50 dual-arm tasks covering diverse skills. We follow the multi-task training setup of (Yuan et al., 2026; Bi et al., 2025; Li et al., 2026): for each task, the training set contains 50 demonstrations in the clean setting and 500 demonstrations in the randomized setting. Models are trained in a multi-task manner with a global batch size of 512. Each method is tested over 100 trials per task in both the clean and randomized settings, and we report task-averaged success rates. We compare against competitive VLA and WAM baselines, including π_0 (Black et al., 2024), $\pi_{0.5}$ (Physical Intelligence et al., 2025), ABot-M0 (Yang et al., 2026), Motus (Bi et al., 2025), LingBot-VA (Li et al., 2026), and Fast-WAM (Yuan et al., 2026). Appendices B and E provide detailed protocols, baseline settings, and per-task success rates.

Results and analysis. Table 1 summarizes the RoboTwin results: AHA-WAM achieves 93.40% success in the clean setting and 92.20% under randomized evaluation, yielding an average success rate of 92.80%. Without robot data pretraining, AHA-WAM improves over Fast-WAM by 0.97 percentage points, showing that our asynchronous design not only increases the closed-loop control frequency but also preserves and improves performance. AHA-WAM also exceeds LingBot-VA, the strongest baseline with large-scale robot-data pretraining, by 0.60 points. The AHA-WAM-Flash variant retains 90.20% average success, incurring only a modest performance drop while further cutting the latency of AHA-WAM by more than half as analyzed in Section 4.5.

Table 1: **RoboTwin 2.0 average success on 50 tasks.**

Method	Robo. P.T.	Clean (%)	Rand. (%)	Avg. (%)
π_0	✓	65.92	58.40	62.16
$\pi_{0.5}$	✓	82.74	76.76	79.75
ABot-M0	✓	81.20	80.40	80.80
Motus from Wan2.2	✗	77.56	77.00	77.28
Motus	✓	88.66	87.02	87.84
LingBot-VA	✓	92.90	91.50	92.20
Fast-WAM	✗	91.88	91.78	91.83
AHA-WAM-Flash	✗	90.48	89.92	90.20
AHA-WAM	✗	**93.40**	**92.20**	**92.80**

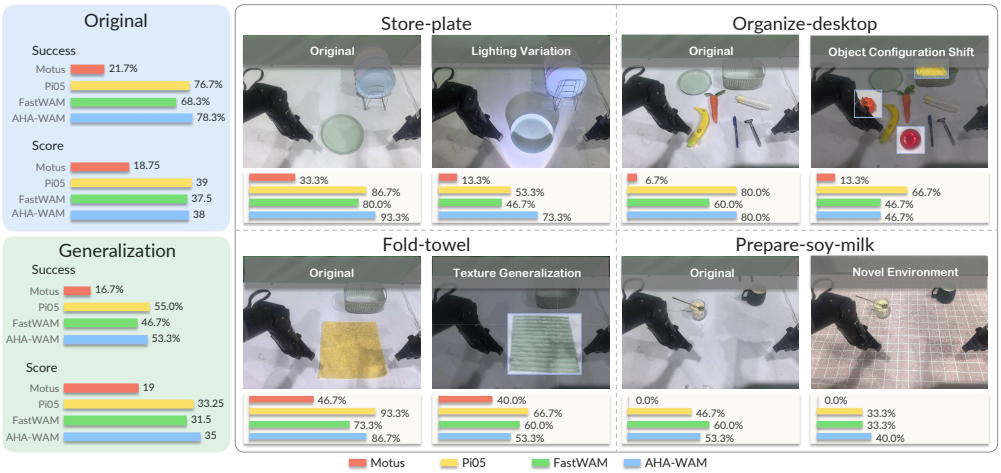

Figure 3: **Real-world task success rates and scores.** Success and 0-3 task scores are computed over 30 trials; scoring criteria are in Appendix C.

4.3 Ablation Studies on RoboTwin

We ablate the components that make asynchronous world-action modeling effective. Naive-Async decouples the video and action branches, but directly reuses the latest planner context without rolling K/V memory or OVCR. We then add each mechanism gradually.

Table 2: **RoboTwin 2.0 ablations.**

Variant	Clean (%)	Rand. (%)	Avg. (%)
Fast-WAM	91.88	91.78	91.83
Naive-Async	88.64	88.56	88.60
+ KV Memory	91.40	90.62	91.01
+ OVCR	91.52	91.42	91.47
AHA-WAM	**93.40**	**92.20**	**92.80**

Table 2 shows that asynchronous execution alone is insufficient. Naive-Async drops from 91.83% to 88.60%, indicating that faster action updates cannot compensate for stale or phase-misaligned planner context. Adding rolling K/V memory recovers performance to 91.01%, showing that persistent planner states help stabilize the reused video context across refreshes. The gain is moderate, which is expected because most RoboTwin tasks are short-to-medium horizon and keep task-relevant objects visible. OVCR further improves success to 91.47%, suggesting that observation-conditioned context adaptation is the more direct remedy for asynchronous planner–executor mismatch. The full AHA-WAM reaches 92.80%, confirming that memory and routing are complementary: memory preserves temporal context, while OVCR aligns it with the current execution state.

4.4 Real-World Robot Experiments

Setup. We evaluate on a bimanual AgileX Piper platform with an ego-view RGB camera, and policies uses only head-view RGB observations, proprioceptive states and language instructions as input. We consider four tasks—*Fold Towel*, *Organize Desktop*, *Prepare Soy Milk*, and *Store Plate*—covering deformable manipulation, long-horizon rearrangement, fine-grained tool use, contact-rich

control, and spatial generalization. For each task, we collect approximately 120 episodes on average. Appendix C provides task details, data scale, execution procedures, and generalization variants.

Implementation. Since Fast-WAM and AHA-WAM are not pretrained on any robot data by default, we pretrain Fast-WAM and AHA-WAM on the selected RoboCOIN subset (Wu et al., 2025), containing 24,600 trajectories and approximately 165 hours of robot data for a fair and stable deployment comparison. Motus and Fast-WAM incur large naive deployment latency, which leads to sparse action updates and unstable closed-loop behavior. We deploy them with an RTC-style non-blocking execution scheme (Black et al., 2026) and action interpolate between predicted chunks, enabling smoother control under high inference latency. AHA-WAM is deployed with its asynchronous planner–executor interface, where the executor performs high-frequency closed-loop updates using the latest observation and reused planner context.

Results. Figure 3 reports real-world performance under original settings and generalization shifts. In the original settings, AHA-WAM achieves 78.33% success, clearly outperforming the WAM baselines Motus (21.67%) and Fast-WAM (68.33%), while matching the strong generalist VLA baseline $\pi_{0.5}$ (76.67%). Under generalization shifts, $\pi_{0.5}$ obtains the highest success rate, while AHA-WAM remains second in success and achieves the highest progress score (35.00). This suggests that, despite not relying on $\pi_{0.5}$-scale generalist pretraining, AHA-WAM attains comparable real-world robustness and delivers the strongest deployment performance among WAM-based baselines. These results indicate that asynchronous world-action modeling improves deployment stability: long-horizon planner context supports task progress, while high-frequency execution enables timely physical correction.

4.5 Latency and Control Frequency

We compare closed-loop inference latency in Tab. 3. Fast-WAM is reported with official latency, while the remainings are measured on a single NVIDIA RTX 5090D GPU. Appendix D gives protocol and optimization details.

Table 3: Inference latency and frequency.

Method	Lat. (ms)	Freq. (Hz)	Speedup
Motus	1866.10	0.54	0.10×
Fast-WAM	190.00	5.26	1.00×
AHA-WAM	41.37	24.17	4.59×
AHA-WAM-Flash	**17.56**	**56.95**	10.82×

Compared with Fast-WAM, AHA-WAM reduces latency from 190.00 ms to 41.37 ms, improving the closed-loop rate from 5.26 Hz to 24.17 Hz. With the distilled sampler, AHA-WAM-Flash further reaches 17.56 ms and 56.95 Hz, yielding a 10.82× speedup while keeping the same asynchronous planner–executor interface.

5 Conclusion

We presented AHA-WAM, an asynchronous horizon-adaptive world-action model that addresses the temporal mismatch between predictive world modeling and reactive action generation. By decoupling a low-frequency video DiT world planner from a high-frequency action DiT executor through an asynchronous modeling interface, AHA-WAM enables reusable predictive context without increasing the action control latency. Experiments on RoboTwin and real-world manipulation tasks demonstrate that asynchronous world-action modeling can improve deployment efficiency while maintaining strong manipulation performance.

Broadly, AHA-WAM suggests a design principle for future WAMs: world prediction and action execution need not share the same temporal horizon. Decoupling slow scalable world modeling from fast reactive control creates a flexible interface where the world model can become more predictive and physically grounded without constraining the control frequency required for real-world deployment.

Limitations and future work. The temporal allocation between planner updates, prediction horizons, and action execution remains task- and embodiment-dependent. Future work could explore adaptive temporal allocation, richer physical world representations, and more challenging long-horizon benchmarks to further evaluate the scalability of asynchronous world-action models.

AI USE STATEMENT

We used ChatGPT to assist with code implementation and manuscript preparation, including language editing and feedback on the clarity of method descriptions and the interpretation of reported results. The authors conducted the experiments, manually checked the code, verified the experimental data, and reviewed the manuscript. The authors take responsibility for the final text, code, results, and conclusions, including all AI-assisted content.

REPRODUCIBILITY STATEMENT

We provide the complete implementation of AHA-WAM in the supplementary materials to facilitate independent reproduction of our method and experiments. We ensure that the submitted code, together with the documented configurations and evaluation protocols, provides a complete and reproducible implementation of AHA-WAM and its experimental pipeline.

REFERENCES

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. *arXiv preprint arXiv:2512.13030*, 2025.

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

Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *NeurIPS*, 2026.

Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *CoRL*, 2023.

Qingwen Bu, Hongyang Li, Li Chen, Jisong Cai, Jia Zeng, Heming Cui, Maoqing Yao, and Yu Qiao. Towards synergistic, generalized, and efficient dual-system for robotic manipulation. *arXiv preprint arXiv:2410.08001*, 2024.

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.

Yilun Du, Sherry Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Josh Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. *NeurIPS*, 2023.

Noriaki Hirose, Catherine Glossop, Dhruv Shah, and Sergey Levine. Asyncvla: An asynchronous vla for fast and robust navigation on the edge. *arXiv preprint arXiv:2602.13476*, 2026.

Yucheng Hu, Yanjiang Guo, Pengchao Wang, Xiaoyu Chen, Yen-Jen Wang, Jianke Zhang, Koushil Sreenath, Chaochao Lu, and Jianyu Chen. Video prediction policy: A generalist robot policy with predictive visual representations. In *ICML*, 2025.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, et al. OpenVLA: An open-source vision-language-action model. In *CoRL*, 2025.

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.

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.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICML*, 2023.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Yang Tian, Sizhe Yang, Jia Zeng, Ping Wang, Dahua Lin, Hao Dong, and Jiangmiao Pang. Predictive inverse dynamics models are scalable learners for robotic manipulation. In *ICLR*, 2025.

Shihan Wu, Xuecheng Liu, Shaoxuan Xie, Pengwei Wang, Xinghang Li, Bowen Yang, Zhe Li, Kai Zhu, Hongyu Wu, Yiheng Liu, et al. RoboCOIN: An open-sourced bimanual robotic data collection for integrated manipulation. *arXiv preprint arXiv:2511.17441*, 2025.

Han Xue, Jieji Ren, Wendi Chen, Gu Zhang, Fang Yuan, Guoying Gu, Huazhe Xu, and Cewu Lu. Reactive diffusion policy: Slow-fast visual-tactile policy learning for contact-rich manipulation. In *ICRA 2025 Workshop*, 2025.

Yandan Yang, Shuang Zeng, Tong Lin, Xinyuan Chang, Dekang Qi, Junjin Xiao, Haoyun Liu, Ronghan Chen, Yuzhi Chen, Dongjie Huo, et al. Abot-m0: Vla foundation model for robotic manipulation with action manifold learning. *arXiv preprint arXiv:2602.11236*, 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.

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*, 2026b.

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.

A Implementation Settings

This appendix records implementation details that complement the experimental setup in Section 4.1. The main text describes the benchmark-level settings and the asynchronous planner–executor schedule; here we summarize the concrete implementation settings used for the RoboTwin runs. We omit logging paths, checkpoint paths, and inactive legacy annealing options.

Model architecture. Table 4 provides the key architecture and hyperparameter settings for AHA-WAM on RoboTwin, including the Video-DiT planner, Action-DiT executor, and OVCR interface. The instantiated model contains approximately 4.99B parameters in the video branch, 1.02B parameters in the action DiT, and 1.22B parameters in the memory and context-routing modules. During asynchronous inference, the action path receives the latest observation and proprioceptive state while querying the most recently available planner context through OVCR.

ODE distillation. Table 5 lists the hyperparameter settings used for ODE-distilled fast sampling. It records the distillation target, teacher–student denoising schedules, and the main optimization settings; dataset paths, checkpoint paths, logging paths, and other run-specific bookkeeping are omitted.

Table 4: Implementation settings for AHA-WAM on RoboTwin.

Component	Subcomponent	Setting
Component settings		
Video-DiT	Backbone	Wan2.2-TI2V-5B video expert
	Observation cameras	3 views: head, left wrist, right wrist
	Image resolution	384×320
	History frames	6
	Video/action frequency ratio	8
	Video RoPE stride	8
	Routed transformer layers	30
	Attention heads / head dimension	24 / 128
Action-DiT	Action horizon / chunk size	64 actions / 16 actions
	Number of action chunks	4
	State / proprio dimension	14
	Action dimension	14
	Training objective	Action prediction with planner-context conditioning
OVCR	Method name	Observation-Guided Video-Context Routing
	Query source	Visual observation context
	Queries per chunk	32
	Target context	Causal video K/V cache
	Granularity	Per action chunk, per transformer layer
Hyperparameter settings		
Training	Optimizer / learning rate / weight decay	AdamW / 1×10^{-4} / 1×10^{-2}
	Learning-rate schedule / warmup	Cosine / first 5% of training
	Global batch size / epochs	512 / 5
	Dataloader workers	16
	Video/action train timesteps	1000
	Video/action train and inference shift	5.0
Inference	Default action denoising steps / CFG	10 / 1.0

Table 5: Hyperparameter settings for ODE-distilled fast sampling.

Item	Setting
Distillation target	Action ODE / flow trajectory
Teacher denoising steps	16
Student denoising steps	2
Distilled timesteps	5,000
Teacher capture schedule	$0, 1, 2, 4, 8, 12, 16$
Prediction parameterization	Flow
Global batch size	512
Learning rate	2×10^{-5}
Optimizer / weight decay	AdamW / 1×10^{-2}
Learning-rate schedule	Cosine
Epochs	5
Local batch size / gradient accumulation	16 / 4
Dataloader workers	16

Training configuration. The optimizer, scheduler, batch-size, epoch, and default inference settings for AHA-WAM are summarized in Table 4, while the corresponding ODE-distillation settings are summarized in Table 5. We use flow matching for both video prediction and action prediction, with noise times sampled from a logit-normal distribution; when both branches are optimized jointly, the video loss and action loss are weighted equally. Horizon-adaptive offset training follows Section 3.3, so each action chunk is trained to consume planner context under randomized planner–executor phase offsets. The distilled AHA-WAM-Flash variant uses the reduced-step sampler described in Appendix D.

B RoboTwin Evaluation Details

This appendix provides the benchmark and baseline details omitted from the main RoboTwin discussion in Section 4.2.

Benchmark protocol. RoboTwin 2.0 is evaluated with the AgileX embodiment and contains 50 dual-arm manipulation tasks spanning a broad range of bimanual skills. Following the multi-task setting used by prior WAM baselines, AHA-WAM is trained with 50 clean demonstrations and 500 randomized demonstrations per task, totaling $2{,}500$ clean and $25{,}000$ randomized demonstrations. Each task is evaluated with 100 clean episodes and 100 randomized episodes, and we report task-averaged success rates. The randomized setting introduces visual and scene-level variations, providing a stronger robustness test than clean evaluation.

Baseline comparison. We compare AHA-WAM with representative VLA and WAM baselines. Fast-WAM is the closest baseline because it also uses a video DiT inside a world-action model, while Motus and LingBot-VA represent unified world-action modeling architectures. π_0, $\pi_{0.5}$, and ABot-M0 provide comparisons to strong generalist VLA policies. When available, we include both embodied-pretrained and Wan2.2-initialized variants to separate the effect of robot-data pretraining from the model architecture. In Table 1, Embodied PT. indicates whether the method uses embodied robot-data pretraining before RoboTwin 2.0 training. We report official published RoboTwin 2.0 results for external baselines when available.

C Real-World Task Execution and Scoring Criteria

Trials and metrics. For each evaluation condition, we conduct $N = 15$ independent trials per task across four real-world manipulation tasks. Each task consists of three ordered milestones, as summarized in Table 6. A trial receives a progress score $q_{j,i} \in \{0, 1, 2, 3\}$ corresponding to the number of completed milestones, where j indexes tasks and i indexes trials. A trial is considered successful only when all milestones are completed ($q_{j,i} = 3$).

For each condition, we compute the aggregate metrics reported in Figure 3 as

$$\text{Success}(\%) = \frac{100}{4N} \sum_{j=1}^{4} \sum_{i=1}^{N} \mathbf{1}[q_{j,i} = 3], \qquad \text{Score} = \frac{1}{4} \sum_{j=1}^{4} \sum_{i=1}^{N} q_{j,i}. \tag{9}$$

Here $\mathbf{1}[\cdot]$ denotes the indicator function. Success is the average success rate over the four tasks. Score measures accumulated partial progress and ranges from 0 to 45 for $N = 15$. Each evaluation condition contains 60 trials per model, and one additional successful trial changes the aggregate success rate by 1.67 percentage points. Figure 4 illustrates the task execution sequences.



Figure 4: Real-world task execution sequences. The bottom row (Spoon Powder) illustrates Prepare Soy Milk. Table 6 defines the three scored milestones for each task; the images show intermediate execution states.

Table 6: Ordered milestone definitions for real-world tasks. Each trial receives 0–3 points according to completed milestones, where 3 indicates full task completion.

Task	Milestone 1	Milestone 2	Milestone 3
Store Plate	Left hand securely grasps the plate.	Plate is transferred to the right hand.	Right hand places the plate in the rack.
Fold Towel	Both hands fold the towel forward.	Right hand folds the towel again.	Right hand places the folded towel in the basket.
Organize Desktop	Left hand places the left-side target objects (e.g., banana and carrot) on the plate.	Right hand places the pen and tool in the basket.	Right hand places the plastic bag roll in the basket.
Prepare Soy Milk	Right hand moves the cup to the workspace center.	Left hand scoops powder with the spoon and pours it into the cup.	Left hand stirs the mixture and returns the spoon.

Distribution shifts. Table 7 summarizes the distribution shifts used for OOD evaluation in Figure 3. Task instructions and completion criteria remain unchanged. Each task contributes 15 OOD trials per model. For tasks with multiple variants, trials are evenly allocated among variants, and all trials are aggregated at the task level before computing the metrics in Eq. 9.

Table 7: Real-world distribution shifts. Trial counts are per model; each task contributes 15 OOD trials.

Task	Change from the original condition	Trials
Store Plate	Room light off; flashlight from above; flashlight from the side. Plate and rack remain unchanged.	$5+5+5$
Organize Desktop	Fewer target objects; replacement target instances; added distractors. The sorting objective remains unchanged.	$5+5+5$
Fold Towel	Replace the original yellow towel with a green towel.	15
Prepare Soy Milk	Cover the original white tabletop with a tablecloth; retain the cup, spoon, and powder.	15

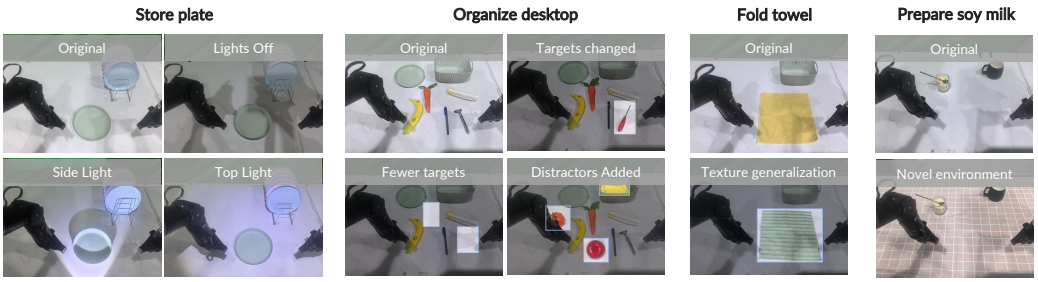

Figure 5: Original real-world conditions and evaluated distribution shifts. Table 7 specifies the trial allocation.

Deployment-side execution of baselines. For Motus and Fast-WAM, we follow their real-time chunk execution protocol with client-side action interpolation. The robot executes the current action chunk while asynchronously requesting the next chunk. When a new chunk arrives, elapsed actions are discarded and overlapping commands are blended with the remaining previous chunk. If the current chunk finishes before a new chunk becomes available, the last valid command is maintained.

This procedure follows the real-time execution protocol of RTC (Black et al., 2026) and requires no policy retraining. It is independent of AHA-WAM's asynchronous planner–executor interface.

D Inference Speedup Details

Measurement protocol. We report L_{chunk}, the end-to-end latency of a single action-chunk inference call. This includes image encoding, incremental planner-context access, OVCR context routing, and Action DiT denoising. Video-DiT prefill runs asynchronously with action inference; we therefore use L_{chunk} as the primary latency metric for closed-loop action frequency and separately track prefill throughput to ensure that planner context can be refreshed in the background. All latency values are measured in milliseconds on RoboTwin 2.0 tasks unless otherwise specified. The first warmup episode is discarded, and remaining per-episode means are averaged. Both baseline and optimized pipelines use bf16 precision, so the reported speedups do not come from reducing numerical precision.

Table 8: Latency notation used in the inference speedup analysis.

Symbol	Definition
L_{chunk}	End-to-end latency of one action-chunk inference
L_{prefill}	Latency of one asynchronous Video-DiT prefill call

CUDA acceleration. Table 9 reports the cumulative optimization path from PyTorch eager inference to the optimized CUDA deployment. The main deployment optimizations are grouped into three categories. First, graph-level static capture compiles backbone modules including the Action DiT, memory/context modules, and VAE encoder into TensorRT engines and replays the fixed denoising loop with CUDA Graphs, reducing Python dispatch and redundant kernel launch overhead. Second, selective `torch.compile` is applied to the Video-DiT prefill path, where full trace export is difficult due to Python-side control flow. Third, hot-path redundancy elimination hoists chunk-level computations outside the denoising loop and skips repeated host-to-device copies for tensors that remain fixed across denoising steps.

Table 9: Cumulative CUDA acceleration ablation on RTX 5090D. CG denotes CUDA Graph.

Stage	Optimization	L_{chunk} (ms)	L_{prefill} (ms)	Runtime
0	PyTorch eager baseline	415.77 ± 0.33	61.15 ± 0.26	A
1	+ Action DiT TRT + CG	83.87 ± 0.51	–	A
2	+ Memory/context TRT + CG	71.37 ± 0.15	–	A
3	+ Video-DiT prefill `compile`	71.45 ± 0.14	34.59 ± 0.08	A
4	+ Hot-path redundancy elimination	50.37 ± 0.27	–	A
5	+ Runtime upgrade (A $\rightarrow$ B)	47.72 ± 0.08	–	B
6	+ TRT video-KV skip-copy	45.77 ± 0.03	–	B
7	+ VAE encoder TRT	41.37 ± 0.03	–	B

Runtime A: PyTorch 2.7.1 + cu128 / Triton 3.3.1 / TensorRT 10.16.1.11. Runtime B: PyTorch 2.12.0 + cu130 / Triton 3.7.0 / TensorRT 10.16.1.11.

Prefill compilation. The goal of prefill optimization is to make planner-context refresh available fast enough for asynchronous deployment. Table 10 compares compile modes. Although `reduce-overhead` gives the lowest prefill latency, it increases L_{chunk}; we therefore use the default mode.

Table 10: Video-DiT prefill compile mode comparison.

Configuration	L_{chunk} (ms)	L_{prefill} (ms)
Eager (no compile)	71.37	57.15 ± 0.14
compile (default)	71.45	34.59 ± 0.08
compile (reduce-overhead)	75.08	25.70 ± 0.22

Hot-path streamlining. The denoising loop contains computations that depend only on chunk-level inputs, such as condition embeddings, positional encodings, and references to planner-context K/V tensors. Hoisting these computations outside the 10-step loop reduces L_{chunk} from 71.45 ms to 63.25 ms. Removing redundant recursive state traversals over modules already in evaluation mode further reduces latency to 50.37 ms. The TensorRT wrapper also reuses static input buffers for video-KV tensors that remain unchanged within a chunk, reducing latency by another 1.95 ms in the 10-step setting.

VAE encoder validation. We validate the VAE encoder TensorRT engine against a PyTorch eager reference using both random inputs and real RoboTwin frames. As shown in Table 11, the TensorRT output remains numerically close to the reference and produces no NaN or Inf values.

ODE distillation. After CUDA acceleration establishes the optimized 10-step path, ODE distillation reduces the number of action denoising steps needed by the sampler. For AHA-WAM-Flash, we freeze the video DiT and distill only the action denoising path so that the student preserves the same planner-context and OVCR interface as the 10-step teacher. The teacher produces a 16-step denoising trajectory, indexed from the noisy state 0 to the final denoised state 16, and we select trajectory anchors $\{0, 1, 2, 4, 8, 12, 16\}$. During training, the student randomly samples a non-final anchor as the starting state and directly predicts the teacher's final denoised action state using a regression loss.

Table 11: VAE encoder numerical validation against PyTorch eager reference.

Metric	Random inputs ($N = 1000$)	Real frames ($N = 1000$)
Cosine similarity (mean)	0.999974	0.999941
Cosine similarity (min)	0.999897	0.999589
Max absolute deviation (min–max)	0.031–0.078	0.023–0.086
Relative max deviation (mean)	1.24%	1.31%
NaN/Inf count	0	0

We sample more frequently from states near the noisy end of the trajectory, because accurate prediction from high-noise states is the key requirement for reducing the number of inference denoising steps.

Table 12: Latency of the CUDA-accelerated action sampler under different ODE-distilled denoising step counts.

Denoising steps	L_{chunk} (ms)	Frequency (Hz)	Latency vs. 10-step
1	14.67	68.3	-64.60%
2	17.56	56.9	-57.50%
4	23.45	42.6	-43.30%
10	41.37	24.2	–

D.1 OVCR Component Ablations

Table 13 examines current-observation conditioning, layerwise routing, residual updates, gating, and the number of routing queries. Relative to the full reference, removing current-observation conditioning reduces average success by 3.87 points, while removing layerwise routing reduces it by 3.31 points. These are the largest decreases among the reported variants, consistent with the importance of conditioning the reused planner context on the current observation at multiple action-DiT layers. The residual-update and gating ablations show smaller decreases of 1.46 and 1.13 points, respectively.

Table 13: Supplementary OVCR ablations on RoboTwin 2.0. The full reference uses $Q = 32$ routing queries. Clean, randomized, and average success rates are reported in percent.

Variant	Clean (%)	Rand. (%)	Avg. (%)
Full reference ($Q = 32$)	93.40	92.20	92.80
Without residual updates	91.24	91.00	91.12
Without gating	91.86	91.04	91.45
$Q = 16$	92.04	91.42	91.73
$Q = 64$	92.82	92.08	92.45
Without layerwise routing	89.74	88.80	89.27
Without current-observation conditioning	89.26	88.16	88.71

Using 16 queries gives 91.73% average success, compared with 92.80% for 32 queries and 92.45% for 64 queries. Thus, increasing the query count beyond 32 provides no improvement in these point estimates, supporting the default choice of $Q = 32$ without requiring twice as many routing queries. The comparison describes the observed configuration sensitivity rather than establishing statistical equivalence between query counts.

D.2 Offset-Training Ablation

Table 14 compares fixed alignment ($\delta = 0$) with the randomized action-start offsets described in Section 3.3. Randomized offsets increase RoboTwin average success from 92.25% to 92.80%, a gain of 0.33 points. This provides a direct ablation of the training choice, with a modest aggregate

improvement in this setting. The randomization covers phases within one action-chunk period; this comparison does not measure robustness to arbitrary planner delays.

Table 14: Fixed versus randomized offset training on RoboTwin 2.0.

Training offsets	Clean (%)	Rand. (%)	Avg. (%)
Fixed ($\delta = 0$)	92.40	92.10	92.25
Randomized ($\delta \sim \mathcal{U}\{0, \ldots, h_a - 1\}$)	93.00	92.60	92.80

D.3 Horizon and Action-Chunk Configuration

Table 15 reports supplementary temporal configurations, where H denotes the video prediction horizon and C denotes the action-chunk length, corresponding to h_v and h_a in Section 3.3. At fixed $C = 16$, increasing H from 32 to 64 improves average success from 90.27% to 92.80%, a gain of 2.31 points. The $H = 64, C = 16$ setting has the highest average success among the evaluated configurations and matches the temporal allocation used by the main model.

Table 15: Sensitivity to video prediction horizon H and action-chunk length C. Values are RoboTwin 2.0 success rates in percent.

H	C	Clean (%)	Rand. (%)	Avg. (%)
16	16	89.52	90.44	89.98
32	32	91.60	91.33	91.47
32	16	91.20	89.33	90.27
64	32	92.80	92.04	92.42
64	16	93.40	92.20	92.80

With the prediction horizon and action-chunk length held fixed, asynchronous execution reduces action-update latency while maintaining competitive task success. Under execution-time perturbations, it further improves recovery performance, supporting the benefit of decoupling planner refreshes from reactive action updates.

D.4 Routing Changes Across Action Chunks

Figure 6 visualizes changes in routing relative to the first observation after a video-DiT prefill, for layers 0, 15, and 29. The plotted total-variation (TV) distance increases over subsequent action chunks at all three layers, showing that the routing changes as newer observations are incorporated while planner context is reused. This diagnostic complements the component ablations by illustrating the temporal variation of the routing mechanism. It does not by itself establish that larger routing changes imply better actions or quantify recovery from external perturbations.

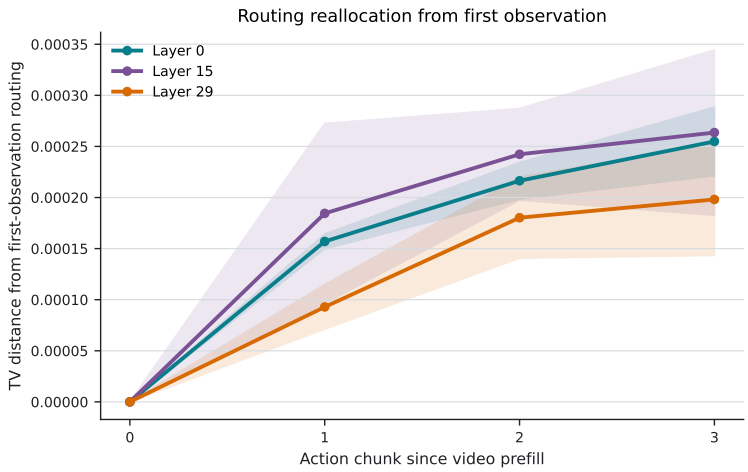

Figure 6: Routing variation across action chunks since video prefill. The curves show TV distance relative to routing from the first observation for layers 0, 15, and 29. Chunk 0 is the reference observation.

E PER-TASK ROBOTWIN SUCCESS RATES

Table 16 reports per-task success rates on RoboTwin 2.0 for AHA-WAM and the selected baselines. All values are percentages, with clean and randomized evaluation reported separately.

Table 16: Per-task success rates on RoboTwin 2.0 under clean and randomized evaluation settings.

Task	AHA-WAM		AHA-WAM-Flash		Fast-WAM		LingBot-VA		$\pi_{0.5}$		Motus	
	Clean	Rand.	Clean	Rand.	Clean	Rand.	Clean	Rand.	Clean	Rand.	Clean	Rand.
Adjust Bottle	100%	100%	100%	100%	100%	100%	90%	94%	100%	99%	89%	93%
Beat Block Hammer	100%	94%	100%	93%	99%	97%	96%	98%	96%	93%	95%	88%
Blocks Ranking RGB	100%	98%	100%	100%	100%	100%	99%	98%	92%	85%	99%	97%
Blocks Ranking Size	98%	98%	97%	91%	94%	98%	94%	96%	49%	26%	75%	63%
Click Alarmclock	100%	100%	100%	100%	100%	100%	99%	100%	98%	89%	100%	100%
Click Bell	100%	100%	100%	100%	100%	100%	100%	100%	99%	66%	100%	100%
Dump Bin Bigbin	99%	92%	95%	95%	97%	96%	89%	96%	92%	97%	95%	91%
Grab Roller	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Handover Block	96%	85%	97%	87%	95%	81%	99%	78%	66%	57%	86%	73%
Handover Mic	99%	99%	94%	92%	99%	100%	94%	96%	98%	97%	78%	63%
Hanging Mug	85%	78%	67%	53%	58%	62%	40%	28%	18%	17%	38%	38%
Lift Pot	100%	100%	100%	100%	100%	100%	100%	99%	96%	85%	96%	99%
Move Can Pot	78%	83%	89%	94%	90%	88%	94%	97%	51%	55%	34%	74%
Move Pillbottle Pad	100%	99%	96%	98%	100%	99%	99%	99%	84%	61%	93%	96%
Move Playingcard Away	100%	100%	98%	100%	100%	100%	100%	99%	96%	84%	100%	96%
Move Stapler Pad	88%	77%	75%	63%	77%	64%	91%	79%	56%	42%	83%	85%
Open Laptop	98%	97%	98%	98%	98%	100%	92%	94%	90%	96%	95%	91%
Open Microwave	40%	46%	9%	23%	62%	45%	82%	86%	34%	77%	95%	91%
Pick Diverse Bottles	89%	85%	83%	87%	80%	85%	89%	82%	81%	71%	90%	91%
Pick Dual Bottles	98%	99%	96%	98%	100%	96%	100%	99%	93%	63%	96%	90%
Place A2B Left	100%	92%	96%	98%	95%	93%	97%	93%	87%	82%	88%	79%
Place A2B Right	96%	92%	98%	96%	93%	99%	97%	95%	87%	84%	91%	87%
Place Bread Basket	89%	94%	89%	98%	91%	93%	97%	95%	77%	64%	91%	94%
Place Bread Skillet	91%	87%	87%	85%	90%	93%	95%	90%	85%	66%	86%	83%
Place Burger Fries	99%	98%	94%	98%	96%	99%	97%	95%	94%	87%	98%	98%
Place Can Basket	81%	94%	83%	79%	71%	69%	81%	84%	62%	62%	81%	76%
Place Cans Plasticbox	100%	100%	98%	100%	99%	96%	100%	99%	94%	84%	98%	94%
Place Container Plate	100%	96%	100%	94%	96%	100%	99%	97%	99%	95%	98%	99%
Place Dual Shoes	91%	99%	92%	89%	94%	88%	94%	89%	75%	75%	93%	87%
Place Empty Cup	100%	100%	100%	98%	100%	100%	100%	100%	100%	99%	99%	98%
Place Fan	100%	87%	96%	94%	96%	96%	99%	93%	87%	85%	91%	87%
Place Mouse Pad	93%	82%	79%	81%	83%	89%	93%	96%	60%	39%	66%	68%
Place Object Basket	81%	80%	89%	85%	89%	88%	91%	88%	80%	76%	81%	87%
Place Object Scale	96%	87%	89%	77%	90%	97%	96%	95%	86%	80%	88%	85%
Place Object Stand	94%	93%	96%	90%	90%	94%	99%	96%	91%	85%	98%	97%
Place Phone Stand	98%	96%	98%	94%	97%	99%	97%	97%	81%	81%	87%	86%
Place Shoe	91%	96%	94%	98%	96%	99%	98%	98%	92%	93%	99%	97%
Press Stapler	98%	99%	98%	100%	90%	97%	85%	82%	87%	83%	93%	98%
Put Bottles Dustbin	83%	96%	83%	77%	95%	90%	87%	91%	84%	79%	81%	79%
Put Object Cabinet	91%	89%	89%	85%	94%	89%	85%	87%	80%	79%	88%	71%
Rotate QRcode	90%	90%	88%	90%	93%	89%	96%	91%	89%	87%	89%	73%
Scan Object	94%	90%	85%	91%	89%	92%	96%	91%	72%	65%	67%	66%
Shake Bottle	100%	100%	100%	100%	100%	100%	100%	97%	99%	97%	100%	97%
Shake Bottle Horizontally	100%	100%	100%	100%	100%	100%	100%	99%	99%	99%	100%	98%
Stack Blocks Three	99%	98%	100%	98%	95%	97%	99%	98%	91%	76%	91%	95%
Stack Blocks Two	100%	99%	100%	100%	100%	100%	100%	98%	97%	100%	100%	98%
Stack Bowls Three	79%	82%	81%	83%	80%	81%	86%	83%	77%	71%	79%	87%
Stack Bowls Two	100%	100%	94%	98%	92%	98%	94%	98%	95%	96%	98%	98%
Stamp Seal	98%	90%	81%	83%	90%	94%	96%	97%	79%	55%	93%	92%
Turn Switch	70%	74%	53%	65%	61%	59%	44%	45%	62%	54%	84%	78%
Average	93.40%	92.20%	90.48%	89.92%	91.88%	91.78%	92.90%	91.50%	82.74%	76.76%	88.66%	87.02%