FASTER-WAM: DO WORLD ACTION MODELS NEED DEEP ACTION MODULES?

Anonymous authors
Paper under double-blind review

ABSTRACT

World Action Models (WAMs) build on pretrained video models, whose representations are grounded in physical dynamics and provide a natural basis for action prediction. Despite this natural foundation, many WAMs still rely on deep, parameter-heavy action-prediction modules that incur high inference latency and may overfit to limited robot demonstrations, restricting their real-world applicability. In this paper, we advocate a world-model-centric principle that concentrates capacity and computation in the video world model, while a lightweight action expert translates the backbone's representations into executable robot actions. We realize this principle through three key choices: Dock of Transformers (DoT) with Lite KV-Fusion to give the shallow, lightweight action expert access to representations from all video layers; world-model-only conditioning of the action expert; and retracted 1D-RoPE for positional alignment between video keys and action queries. We test this principle using Faster-WAM, a world-model-centric WAM with only a single-layer action expert. Despite this restriction on action-specific computation, Faster-WAM achieves competitive control performance on LIBERO and RoboTwin 2.0 without additional embodied pretraining. It provides approximately $3.7\times$ and $1.3\times$ inference speedups over Fast-WAM and $\pi_{0.5}$, respectively. Consistent with its world-model-centric design, Faster-WAM demonstrates stronger generalizability under distribution shifts: the same LIBERO-trained policy achieves 78.3% success on LIBERO-Plus, exceeding Fast-WAM and LingBot-VA by 26.8 and 8.8 percentage points, respectively. Finally, real-robot experiments demonstrate success rates comparable to Fast-WAM, with substantially lower inference latency and shorter task-completion times.

1 INTRODUCTION

World Action Models (WAMs) and Vision-Language-Action (VLA) models are two major paradigms for bridging pretrained foundation models and robot control. VLAs typically adapt pretrained vision-language models for control (Black et al., 2025b;a), whereas WAMs build on pretrained video models (Kim et al., 2026). By modeling how objects move and interact across space and time, video models learn representations grounded in physical dynamics, providing a natural basis for action prediction.

Despite their natural foundation, many WAMs still devote substantial computation and capacity to action prediction. Shared-backbone architectures use the large video backbone for action prediction (Kim et al., 2026; Li et al., 2026a)[1], whereas Mixture-of-Transformers (MoT) architectures introduce separate, slimmer action experts to reduce action-specific computation and capacity (Bi et al., 2026; Yuan et al., 2026b). In both architectures, however, action prediction still relies on deep, parameter-heavy modules. This raises two concerns. First, repeated action denoising through large modules incurs high inference latency (Fig. 1b), thereby limiting real-world applicability (Appx. G.2). Second, robot demonstrations are relatively limited compared with video-pretraining corpora. These high-capacity action-prediction modules may therefore overfit to the limited robot demonstrations, weakening the generalizability afforded by large-scale video pretraining. These observations raise a

[1]We classify LingBot-VA as a shared-backbone architecture based on its released implementation, in which video and action tokens traverse the same transformer blocks.

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.

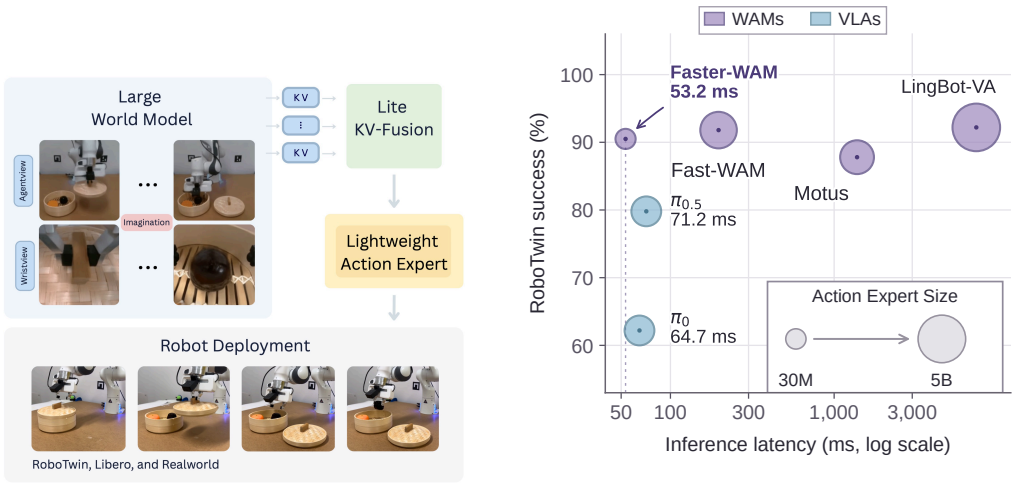


Figure 1: (a) Faster-WAM (left) connects a pretrained video world model to a single-layer action head through Lite KV-Fusion, which learns softmax-weighted mixtures of key–value representations across all backbone layers. The fused representations condition action generation for robot control. (b) Circle size denotes decoder parameters (compressed scale); LingBot-VA counts its shared 5B backbone. Latency measurements for the RoboTwin configuration are reported in Tab. 2.

central question: *can the pretrained video world model support effective control with substantially less action-specific computation?*

To answer this question, we present **Faster-WAM**, a *world-model-centric* WAM. The model concentrates capacity and computation for physical understanding and reasoning in its video backbone. A lightweight action expert then translates the backbone's representations into robot actions. We realize this design through three key choices:

(1) Dock of Transformers (DoT) architecture. The proposed DoT architecture connects a lightweight action expert to the video backbone through *Lite KV-Fusion*, which learns, for each action layer, to fuse the native key–value (KV) pairs cached across all video layers. This decouples the action expert's depth from its access to video-layer representations, which are coupled in MoT architectures where each action layer can only access its corresponding video layer (Yuan et al., 2026b; Ma et al., 2026). Learning the fusion removes the need to manually select extraction layers, and reusing the native KV cache avoids the additional dense KV reprojection used in prior works.

(2) World-model-only conditioning. Following the world-model-centric principle, we condition the action expert solely on the video backbone's representations, rather than also providing direct language and proprioceptive inputs as in some prior WAMs (Yuan et al., 2026b; Cai et al., 2026a). Routing these inputs only through the world model may reduce over-reliance on them and overfitting to limited robot demonstrations, helping preserve the generalizability afforded by large-scale video pretraining.

(3) RoPE axis-frequency alignment. Because Lite KV-Fusion reuses native video keys, we introduce *retracted 1D-RoPE* to align their rotary positional encodings (Su et al., 2024) with those of the action queries. It removes the fused video keys' original 3D spatiotemporal rotation and re-encodes them using the action queries' 1D temporal axis and frequency grid. This addresses axis and frequency mismatches in some prior WAMs, which combine action queries with video-derived keys in joint attention without explicit rotary realignment (Yuan et al., 2026b; Ye et al., 2026a; Cai et al., 2026a). Together, these design choices instantiate the world-model-centric principle in Faster-WAM.

Our experiments further support this world-model-centric design. Even without additional embodied pretraining and with a single-layer action expert, Faster-WAM achieves competitive control performance on LIBERO (Liu et al., 2023) and RoboTwin 2.0 (Chen et al., 2026b). It provides approximately $3.7\times$ and $1.3\times$ inference speedups over Fast-WAM (Yuan et al., 2026b) and $\pi_{0.5}$ (Black et al., 2025a), respectively. To benchmark generalizability under distribution shifts, we evaluate the same LIBERO-trained policy on LIBERO-Plus (Fei et al., 2026), where it outperforms Fast-WAM by 26.8 percentage points. In real-robot experiments, Faster-WAM achieves success rates comparable to those of existing WAMs, with shorter task-completion times. Finally, our ablation studies trace the progression toward Faster-WAM and compare alternative fusion and RoPE designs (Fig. 3).

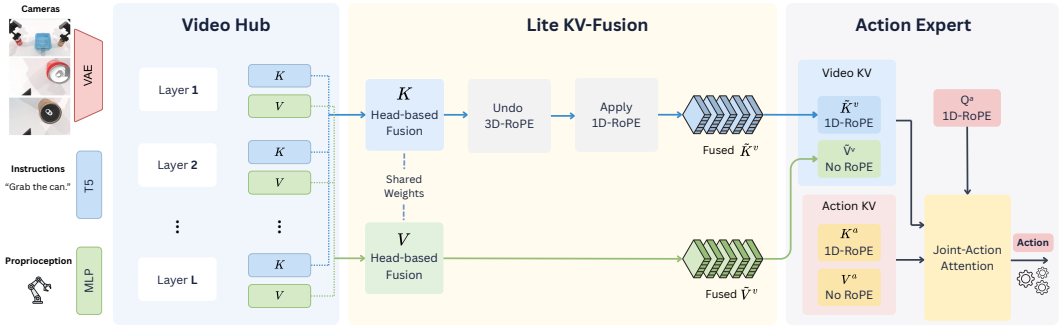


Figure 2: Faster-WAM. Lite KV-Fusion connects a shallow action expert to representations across the full video backbone, forming the Dock of Transformers architecture. Language and proprioception condition the video backbone, while the action expert receives these inputs through the fused video representations. Retracted 1D-RoPE aligns the fused video keys with the action queries before joint attention.

2 METHOD

At each replanning step, Faster-WAM predicts an action chunk—a sequence of future robot actions—from camera observations, a language instruction, and the robot's proprioceptive state. It follows a *world-model-centric* design: the pretrained video backbone provides the main capacity for physical understanding and reasoning, while a lightweight action expert translates its representations into robot actions. Specifically, we use the pretrained Wan2.2-5B video DiT (Wang et al., 2025) with a single-layer action DiT. Figure 2 illustrates the three design choices described below: **Dock of Transformers**, **world-model-only conditioning**, and **RoPE axis–frequency alignment**.

2.1 DOCK OF TRANSFORMERS (DOT)

Video pretraining provides representations grounded in physical dynamics, offering a natural basis for action prediction (Ye et al., 2026b). To effectively use these representations with a shallow, lightweight action expert, we need an interface that decouples the action expert's depth from its access to video-layer representations. However, MoT architectures typically couple these two aspects, so reducing action-expert depth can restrict access to some video layers. We therefore introduce **Lite KV-Fusion**, an interface within DoT that learns, for each action layer, to fuse the native key–value (KV) pairs cached across all video layers.

Lite KV-Fusion. Let L_v and L_a denote the depths of the video backbone and action expert, respectively. Both modules use H attention heads of dimension D, enabling direct KV reuse. For video layer ℓ and head h, let $K^v_{\ell h}, V^v_{\ell h} \in \mathbb{R}^{S_v \times D}$ denote the native cached keys and values for S_v video tokens. The cached keys have already undergone the backbone's key normalization and 3D RoPE.

Each action layer j learns a separate distribution over video layers for each attention head:

$$\beta_{jh\ell} = \frac{\exp(\alpha_{jh\ell})}{\sum_{r=1}^{L_v} \exp(\alpha_{jhr})}, \qquad \widehat{K}^v_{jh} = \sum_{\ell=1}^{L_v} \beta_{jh\ell} K^v_{\ell h}, \qquad \widehat{V}^v_{jh} = \sum_{\ell=1}^{L_v} \beta_{jh\ell} V^v_{\ell h}. \tag{1}$$

Here, $\alpha_{jh\ell}$ are learnable logits, and $\beta_{jh\ell}$ is the normalized weight assigned to video layer ℓ by head h of action layer j. Since keys and values are strictly paired in Transformers, we fuse them using the same mixing weights. We initialize all logits $\alpha_{jh\ell}$ to zero, giving each video layer equal weight, and optimize them jointly with the policy. Learning the fusion removes the need to manually select extraction layers (Ma et al., 2026) while giving the shallow, lightweight action expert access to representations from all video layers.

By reusing native KV pairs, Lite KV-Fusion avoids the additional dense projections used by interfaces that construct keys and values from backbone hidden representations (Li et al., 2025; Chen et al., 2026a). It adds HL_v trainable parameters per action layer. Since Faster-WAM uses a single-layer

action expert, Lite KV-Fusion introduces only 720 trainable parameters in total. The fusion costs $\mathcal{O}(L_v HD)$ per token, compared with $\mathcal{O}(H^2D^2)$ for a dense projection between HD-dimensional representations.

Joint action attention. The action expert integrates the fused video representations through joint attention. Using normalized and positionally encoded action queries and keys, we compute

$$A_{jh} = \mathrm{Softmax}\Big(Q^a_{jh}[K^a_{jh}; \widehat{K}^v_{jh}]^\top/\sqrt{D} + M\Big), \qquad O^a_{jh} = A_{jh}[V^a_{jh}; \widehat{V}^v_{jh}], \tag{2}$$

where $[\,\cdot\,;\,\cdot\,]$ denotes concatenation along the token dimension and M is an additive attention mask that controls access to future video tokens when present. The masking constraints are specified in Sec. 2.4.

Together, the shallow, lightweight action expert and Lite KV-Fusion form the **Dock of Transformers** (DoT) architecture. Each action layer can access representations from all video layers, decoupling action-expert depth from access to video-layer representations. This allows L_a to be chosen independently of L_v.

Compatibility with QK-Norm and RoPE. Lite KV-Fusion operates directly on cached keys after any QK-Norm (Henry et al., 2020) and RoPE (Su et al., 2024) transformations applied by the video backbone. QK-Norm stabilizes attention by constraining query and key norms, and Lite KV-Fusion preserves the upper bound on key norms (Prop. 2.1). Its head-specific scalar weights also commute with the shared rotary transform, preserving RoPE positional encoding (Prop. 2.2).

Proposition 2.1 (Norm-bound preservation in Lite KV-Fusion)**.** *For each token position and attention head, the Euclidean norm of the fused key does not exceed the largest norm among its input keys. The proof is provided in Appx. B.1.*

Proposition 2.2 (RoPE preservation in Lite KV-Fusion)**.** *If all source layers share the same rotary transform at each token position, fusing their rotated keys is equivalent to applying that transform after fusing their unrotated keys. The proof is provided in Appx. B.2.*

In particular, Prop. 2.2 permits the exact removal of the native video rotation after fusion, enabling the positional alignment introduced in Sec. 2.3.

2.2 World-Model-Only Conditioning

Following the world-model-centric principle, we route all conditioning to the action expert through the video backbone. Existing WAMs typically condition both modules directly on language tokens and proprioceptive states via cross-attention. We remove these side inputs and cross-attention from the action expert, forcing it to obtain all visual, linguistic, and proprioceptive context from the video world model. This design reduces the risk of over-reliance on side inputs and grounds action prediction on representations encoded by the video model, which is typically more generalizable owing to the large-scale video pretraining. It also further lightens the action expert and removes the corresponding cross-attention computation from each action denoising step.

2.3 RoPE Axis–Frequency Alignment

In most MoT-based WAMs, The video backbone uses 3D spatiotemporal RoPE, while the action expert uses 1D temporal RoPE. However, several works combine action queries with video-derived keys in joint attention without explicit rotary realignment (Yuan et al., 2026b; Ye et al., 2026a; Cai et al., 2026a). With the native RoPE encodings, action queries and video keys use different coordinate axes and frequency grids. Even the temporal frequency grids differ: video RoPE assigns only part of each head to time, while action RoPE uses the full head.

Proposition 2.3 (RoPE axis–frequency mismatch)**.** *With native 1D action and 3D video RoPE, action indices enter the relative phases of spatial rotary pairs, while mismatched temporal frequencies introduce dependence on absolute indices. The proof is provided in Appx. B.3.*

Therefore, we introduce *retracted 1D-RoPE* to align the fused video keys with the action queries. For a video token at $p = (t, y, x)$, let $R^v_h(p)$ denote its native 3D rotary transform and $R^a_h(t)$ the

action head's 1D transform, with t expressed on the action temporal scale. For each action layer j, we compute

$$k_{jh}^{v\to a}(p) = R_h^a(t)\, R_h^v(p)^\top\, \widehat{k}_{jh}^v(p), \tag{3}$$

before the action joint attention. This removes the original 3D rotation via inverse rotation and re-encodes the key using the action queries' temporal axis and frequency grid. The resulting keys will be used instead of $\widehat{K}_{jh}^v$ in Eq. (2); the fused values remain unchanged. Appx. B.4 provides the derivation.

In Faster-WAM, action tokens attend only to the observed frame at temporal position $t = 0$, during both training and inference. Since $R_h^a(0) = I$, reapplication reduces to the identity. The video backbone retains its native 3D RoPE whose content can still encode spatial information; retraction removes the explicit rotary factor only from the fused keys supplied to the action expert.

2.4 Training and Inference

During training, we jointly optimize the video backbone, action expert, and Lite KV-Fusion. The video backbone receives gradients from both video and action flow-matching losses, while the action expert receives gradients only from the action loss. The video backbone does not attend to action tokens, while the action expert jointly attends to action tokens and fused video representations. Attention masks prevent observed-frame and action tokens from attending to future video tokens. At inference, we prefill the video backbone's KV cache once per action chunk from the current observation. Lite KV-Fusion aggregates this cache into a fixed context that the action expert reuses across denoising steps. Further training and inference details are provided in Appx. C and E.

2.5 Relationship to Related Work

Several recent and concurrent works aim to reduce inference latency of WAMs.

- *Fast-WAM* (Yuan et al., 2026b) eliminates future-video generation by prefilling the observed frame. Faster-WAM inherits this scheme and further reduces action-side computation, achieving a 3.7–3.8× speedup over Fast-WAM.
- *Light-WAM* (Li et al., 2026b) pairs a smaller video backbone with single-pass action prediction. It reports 76.4% success on RoboTwin, compared with Fast-WAM's 91.9%, a difference of 15.5 percentage points.
- *Concurrent work* by Zhao et al. (2026) retains future-latent conditioning through a single video-expert pass and reduces repeated computation through sparse video–action interaction and interval KV fusion. Their method retains a deep action expert with sparse access to future-conditioned representations, whereas Faster-WAM uses a single-layer action expert conditioned on observed-frame representations. Relative to Fast-WAM, our method achieves a 3.7–3.8× inference speedup, compared with approximately $1.27\times$ for Zhao et al. (2026) under their respective measurement settings.

More comprehensive discussion of related work is provided in Appx. A.

3 Experiments

We revisit the central question of this work: *can the pretrained video world model support effective control with substantially less action-specific computation?*

To answer it, we evaluate an extreme case of the proposed design: Faster-WAM, a world-model-centric WAM with only a single-layer action expert. We focus on two questions:

1. Can Faster-WAM retain competitive control performance while reducing inference latency?
2. Does its world-model-centric design improve generalizability under distribution shifts?

We evaluate control performance and inference latency on LIBERO and RoboTwin 2.0. To assess generalizability under distribution shifts, we evaluate the same LIBERO-trained policy on LIBERO-Plus. We further examine whether these benefits extend to real-robot control and use ablation studies to assess the contributions of the three design choices. Lastly, we evaluate the world-model-centric design with an alternative video backbone in Appx. D.3.

3.1 Competitive Control with Lower Inference Latency

To address the first question, we evaluate Faster-WAM on LIBERO (Liu et al., 2023) (Tab. 1) and RoboTwin 2.0 (Chen et al., 2026b) (Tab. 2), following the established protocols of prior WAMs (Bi et al., 2026; Yuan et al., 2026b; Li et al., 2026a). We report task success rate and model-computation latency per action chunk. The complete dataset, training, evaluation, and latency protocols are provided in Appx. D and E.

Table 1: LIBERO success rates (%) and inference latency per action chunk. Baseline success rates are taken from the respective papers. Reported mean $\pm$ std are over three evaluation runs. Latency measurement is taken on a 24 GiB consumer-level GPU. ET denotes pretraining on robot data.

Method	Type	ET	Spatial	Object	Goal	Long	Average	Latency (ms) ↓	Relative latency
π_0 (Black et al., 2025b)	VLA	✓	96.8	98.8	95.8	85.2	94.1	65.0	1.3×
$\pi_{0.5}$ (Black et al., 2025a)	VLA	✓	98.8	98.2	98.0	92.4	96.9	67.7	1.3×
X-VLA (Zheng et al., 2026)	VLA	✓	98.2	98.6	97.8	97.6	98.1	94.1	1.8×
LingBot-VA (Li et al., 2026a)	WAM	✓	98.5	99.6	97.2	98.5	98.5	4756.9‡	91.4×‡
Motus (Bi et al., 2026)	WAM	✓	96.8	99.8	96.6	97.6	97.7	–	–
GE-ACT (Liao et al., 2026)	WAM	✓	98.2	97.6	95.8	94.4	96.5	200.0*	3.8×*
Fast-WAM (Yuan et al., 2026b)	WAM	✗	98.2	100.0	97.0	95.2	97.6	196.0‡	3.8×‡
Faster-WAM (Ours)	WAM	✗	99.1 ± 0.3	99.5 ± 0.1	98.4 ± 0.2	96.2 ± 0.3	98.3 ± 0.1	52.0	1.0×

* Latency reported in (Liao et al., 2026), measured on the same GPU model.
‡ Fast-WAM and LingBot-VA latencies combine separately measured components as full pipelines exceed the GPU memory. LingBot-VA timing is based on its released variant.

Table 2: RoboTwin 2.0 success rates (%) and inference latency per action chunk. Baseline success rates are taken from the respective papers. Reported mean $\pm$ std are over three evaluation runs. ET denotes embodied pretraining on robot data.

Method	Type	ET	Clean	Rand.	Average	Latency (ms) ↓	Relative Latency
π_0 (Black et al., 2025b)	VLA	✓	65.9	58.4	62.2	64.7	1.2×
$\pi_{0.5}$ (Black et al., 2025a)	VLA	✓	82.7	76.8	79.8	71.2	1.3×
Motus (Bi et al., 2026)	WAM	✓	88.7	87.0	87.8	1380.3	25.9×
LingBot-VA (Li et al., 2026a)	WAM	✓	92.9	91.5	92.2	7386.1	138.8×
Motus from Wan2.2 (Bi et al., 2026)	WAM	✗	77.6	77.0	77.3	–	–
LingBot-VA from Wan2.2 (Li et al., 2026a)	WAM	✗	80.6	–	–	–	–
Fast-WAM (Yuan et al., 2026b)	WAM	✗	91.9	91.8	91.8	196.8	3.7×
Faster-WAM (Ours)	WAM	✗	90.4 ± 0.3	90.6 ± 0.2	90.5 ± 0.2	53.2	1.0×

Competitive Control Performance Despite restricting the action expert to a single layer, Faster-WAM achieves $98.3 \pm 0.1\%$ average success on LIBERO and $90.5 \pm 0.2\%$ on RoboTwin without additional embodied pretraining. Across both benchmarks, Faster-WAM remains competitive with existing VLAs and WAMs, including those that use additional embodied pretraining.

Faster Inference Speed We measure the inference latency of Faster-WAM and the baselines on the same machine with a 24 GiB consumer GPU. Unless otherwise specified, measurements include encoding, conditioning, and action generation. The complete profiling protocol and results are provided in Appx. E.

Faster-WAM and Fast-WAM use the same action-generation protocol, predicting 32-action chunks with 10 denoising steps. Under this matched setting, Faster-WAM reduces inference latency from 196.0 to 52.0 ms on LIBERO and from 196.8 to 53.2 ms on RoboTwin, corresponding to approximately 3.8× and 3.7× speedups, respectively. Faster-WAM also achieves lower inference latency than representative VLAs such as π_0 and $\pi_{0.5}$ under their native configurations. The phase-wise breakdown in Appx. E (Fig. 5) shows that the main latency reduction comes from action generation. Its latency decreases from approximately 150 ms in Fast-WAM to 6 ms in Faster-WAM, while encoding and conditioning costs remain similar. Lower inference latency supports responsive real-world control by reducing the delay between observation and action prediction as the robot and its environment evolve.

Together, these results answer the first question: Faster-WAM, with its single-layer action expert, preserves access to the video hierarchy through DoT, yielding competitive control at substantially lower latency.

3.2 World-Model-Centric Design Improves Generalization

To answer our second question, we evaluate Faster-WAM on LIBERO-Plus (Fei et al., 2026) under an out-of-distribution (OOD) setting. The model is trained on LIBERO and evaluated across seven perturbation categories in LIBERO-Plus. Complete evaluation details are provided in Appx. D.2.2.

Table 3: LIBERO-Plus success rates (%) under seven perturbation categories. Faster-WAM results are reported as mean $\pm$ std over three evaluation runs. Baseline results are taken from Zhang et al. (2026); Huang et al. (2026). ET denotes embodied pretraining.

Method	ET	Background	Camera	Language	Light	Object	Robot	Sensor	Overall
π_0 (Black et al., 2025b)	✓	81.4	13.8	58.8	85.0	68.9	6.0	79.0	53.6
$\pi_{0.5}$ (Black et al., 2025a)	✓	94.6	75.4	85.6	96.9	85.7	77.5	89.7	85.7
X-VLA (Zheng et al., 2026)	✓	96.0	23.4	75.7	88.2	71.8	89.7	62.7	71.4
LingBot-VA (Li et al., 2026a)	✓	53.1	40.9	86.4	82.3	76.2	83.0	64.4	69.5
GE-ACT (Liao et al., 2026)	✓	86.0	60.7	77.4	95.8	80.2	77.0	90.9	80.3*
Fast-WAM (Yuan et al., 2026b)	✗	53.7	16.4	68.9	78.2	60.7	44.5	37.7	51.5
Faster-WAM (Ours)	✗	60.7 ± 0.5	76.5 ± 0.6	90.9 ± 0.6	94.2 ± 0.7	81.7 ± 0.1	58.3 ± 0.3	84.6 ± 0.5	78.3 ± 0.1

* GE-ACT uses separate suite-specific checkpoints; Faster-WAM uses one checkpoint shared across all four suites.

Faster-WAM and Fast-WAM build on the same pretrained video world model, Wan2.2-5B (Wang et al., 2025), and perform comparably on standard LIBERO. On LIBERO-Plus, Faster-WAM achieves 78.3% overall success, exceeding Fast-WAM by 26.8 percentage points and outperforming it across all seven perturbation categories (Tab. 3). Without additional embodied pretraining or data augmentation, Faster-WAM also outperforms π_0, X-VLA, and LingBot-VA, trailing only GE-ACT and $\pi_{0.5}$ among the compared methods. The GE-ACT result is not directly comparable because it uses separate suite-specific checkpoints, whereas Faster-WAM uses one checkpoint shared across all four suites. This broad improvement suggests that concentrating capacity in the video backbone better preserves the generalization benefits of the large-scale pretrained video world model.

3.3 Additional Study

3.3.1 Real-World Control with Faster-WAM

We next evaluate whether Faster-WAM's favorable performance–latency trade-off transfers to physical control. Using a Franka Panda arm, we consider four tasks spanning pick-and-place, occluded sequential manipulation, contact-sensitive pushing, and long-horizon object arrangement. Each policy is jointly trained on 100 demonstrations per task and evaluated over 20 rollouts per task, except LingBot-VA on `dish_utensil`, for which we use 10 rollouts due to its substantially higher inference latency. Complete experimental details are provided in Appx. F.

Table 4: **Real-world performance on the Franka Panda arm.** Average completion time and average inference latency per action chunk are measured over successful rollouts on the deployment workstation.

Method	ET	place_cube		pick_fruit		push_T		dish_utensil		Avg. SR (%) ↑	Latency (ms) ↓
		SR (%) ↑	CT ↓	SR (%) ↑	CT ↓	SR (%) ↑	CT ↓	SR (%) ↑	CT ↓		
$\pi_{0.5}$	✓	100	10.8	100	24.4	55	22.8	90	39.0	86.3	151
LingBot-VA	✓	100	117.8	90	175.3	10	112.5	10	269.0	52.5	10,897
Fast-WAM	✗	95	21.9	95	36.7	70	24.6	60	85.0	80.0	502
Faster-WAM	✗	100	18.2	90	35.1	65	23.8	70	72.0	81.3	129

SR: Success rate (%) **CT**: Completion time (s) **ET**: Embodied pretraining

As shown in Tab. 4, without embodied pretraining, Faster-WAM achieves an average success rate of 81.3%, comparable to Fast-WAM's 80.0%. The embodied-pretrained $\pi_{0.5}$ attains the highest

measured success rate of 86.3%. LingBot-VA achieves 52.5% under the same downstream setting: it performs strongly on the two pick-and-place tasks but drops to 10% on both the contact-sensitive `push_T` and long-horizon `dish_utensil` tasks. We follow LingBot-VA's released training code (Li et al., 2026a).

On the deployment workstation, Faster-WAM reduces inference latency from Fast-WAM's $502\,\mathrm{ms}$ to $129\,\mathrm{ms}$ per action chunk, a $3.9\times$ speedup, and is slightly faster than $\pi_{0.5}$ at $151\,\mathrm{ms}$. LingBot-VA requires $10{,}897\,\mathrm{ms}$ per chunk, which limits its feasibility for real-world robot tasks requiring frequent replanning. Although these absolute values are not directly comparable to the controlled profiling results in Sec. 3.1, the speedup over Fast-WAM remains consistent ($3.9\times$ versus $3.8\times$).

Faster-WAM also achieves lower mean completion times than all evaluated WAM baselines on all four tasks. Despite its slightly higher per-chunk inference latency, $\pi_{0.5}$ achieves shorter mean completion times than Faster-WAM on all four tasks. These results distinguish inference efficiency from overall task execution speed: faster policy calls do not necessarily yield shorter completion times.

Together, these results show that Faster-WAM retains competitive real-world task success with a single-layer action expert while reducing both inference latency and mean completion time relative to the evaluated WAM baselines. Illustrative real-world rollouts are provided in Appx. G.

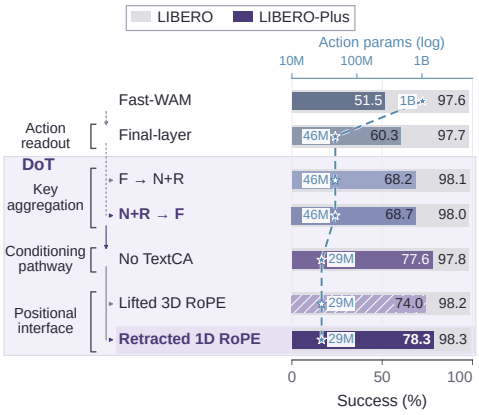


Figure 3: Steps towards Faster-WAM. Branches show alternatives, boldface marks the adopted choices, and stars indicate action-head parameter counts. F: fusion; N: QK-Norm; R: RoPE.

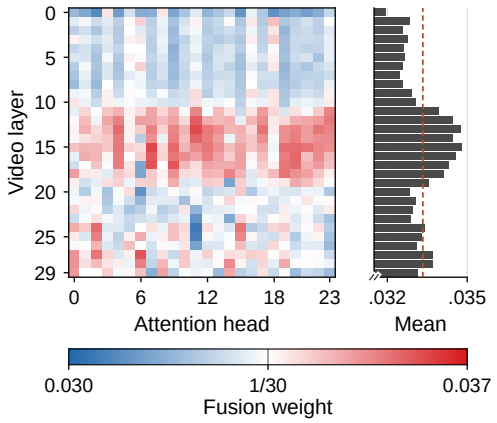


Figure 4: Learned KV-fusion weights by layer and head (left) of Faster-WAM on Libero, with the mean across heads (right). White and the dashed line mark $1/30$.

3.3.2 Ablating the World-Model-Centric Design

LIBERO measures in-distribution control, while LIBERO-Plus perturbs the same task suites to probe OOD generalization. Fig. 3 traces how each design choice affects these two regimes. A single-layer action expert already improves OOD generalization over Fast-WAM, even when it accesses only the final video layer, while preserving in-domain control. Lite KV-Fusion further improves both regimes by exposing the shallow expert to the full hierarchy of world-model representations. We compare two variants: F→N+R fuses keys before QK-Norm and RoPE and then applies QK-Norm and RoPE after fusion, while N+R→F directly fuses the processed keys. They perform comparably, with a small OOD advantage for N+R→F. Because this variant also directly reuses the backbone's native KV cache, we adopt it thereafter.

We next remove action-side text cross-attention (TextCA) and direct proprioceptive conditioning, requiring the action expert to obtain all context through the video world model. This produces a substantial OOD gain with a small in-domain drop. We hypothesize that the mismatch between the video and action RoPE bases becomes more consequential once precise proprioceptive information must pass through the video representations. Both alignment methods recover the in-domain performance, consistent with this explanation. Lifted 3D RoPE, however, loses part of the OOD gain, possibly because assigning pseudo spatial anchors to action tokens introduces an additional spatial prior. Retracted 1D-RoPE avoids this assumption and retains stronger OOD generalization; we therefore adopt it in Faster-WAM.

3.3.3 VISUALIZATION OF LITE KV-FUSION

Fig. 4 visualizes the layer weights learned by Lite KV-Fusion on LIBERO. The learned mixtures are non-uniform and vary across attention heads, with different heads emphasizing different depths of the video backbone. Together with the improvement over final-layer readout in Fig. 3, this suggests that Lite KV-Fusion draws complementary information from the video hierarchy instead of collapsing to a single extraction layer. It provides this adaptive readout with only 720 mixing logits, avoiding manual layer selection as used in DiT4DiT (Ma et al., 2026).

4 LIMITATIONS

Our experiments do not include additional embodied pretraining. Whether it further improves control performance and generalizability under the world-model-centric design remains to be investigated. We validate DoT with two video world models, Wan2.2 and LTX-Video. Its applicability to newer video world models, such as Wan3.0-Video, and to non-generative predictive video models, such as V-JEPA 2 (Assran et al., 2025), remains to be explored. We evaluate Faster-WAM as an extreme case with a single-layer action expert, without conducting an exhaustive search for the optimal depth. Our results support the world-model-centric principle by showing that competitive control does not necessarily require deep, parameter-heavy action experts. They do not establish that a single layer is optimal or that increasing depth is detrimental. Further study is needed to understand how action-expert depth affects control performance, generalizability, and inference latency. Finally, our physical evaluation is limited to one robot platform and four manipulation tasks. Broader evaluation across robot embodiments, task families, and deployment conditions is needed to assess how well DoT generalizes to other real-world settings.

5 CONCLUSION

We asked whether World Action Models need deep action modules. Our results suggest that they need not: action-expert depth can be decoupled from access to the video hierarchy, allowing the pretrained world model to provide the primary capacity for physical understanding and reasoning. Following this world-model-centric principle, Faster-WAM uses a single-layer action expert. Its DoT architecture employs Lite KV-Fusion to access representations across all video layers, while world-model-only conditioning and retracted 1D-RoPE route and align contextual information through the video backbone.

Without additional embodied pretraining, Faster-WAM retains competitive control performance on LIBERO and RoboTwin 2.0 while reducing matched inference latency by 3.7–$3.8\times$ over Fast-WAM. The same LIBERO-trained policy improves success on LIBERO-Plus by 26.8 percentage points, and real-robot experiments further show competitive success with lower online latency and shorter completion times. These findings suggest that making better use of pretrained world-model representations can support efficient and generalizable control with little action-specific computation.

AI USE STATEMENT

In this work, we used generative AI tools for < formulating mathematical claims, assisting in the writing of proofs, implementation of methods >. We have not used generative AI tools for other tasks with required disclosure, and the rest of the required disclosure tasks are not applicable to this work. Additionally, we used generative AI tools for <creating or modifying scientific figures, brainstorming, editing a research paper to improve readability>. We have reviewed all AI-assisted work. We checked the correctness of the mathematical claims and proofs. LLM-generated code was verified and tested for correctness by authors. The LLM-generated figures were verified. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Mahmoud Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, et al. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, et al. Motus: A unified latent action world model. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2026.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proc. Conf. Robot. Learn.*, 2025a.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, et al. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Proc. Robot.: Sci. Syst.*, 2025b.

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

Junhao Cai, Zetao Cai, Jiafei Cao, Yilun Chen, Zeyu He, Lei Jiang, Hang Li, Hengjie Li, Yang Li, Yufei Liu, et al. Internvla-a1: Unifying understanding, generation and action for robotic manipulation. *arXiv preprint arXiv:2601.02456*, 2026b.

Jun Cen, Chaohui Yu, Hangjie Yuan, Yuming Jiang, Siteng Huang, Jiayan Guo, Xin Li, Yibing Song, Hao Luo, Fan Wang, et al. Worldvla: Towards autoregressive action world model. *arXiv preprint arXiv:2506.21539*, 2025.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Proc. Int. Conf. Mach. Learn.*, 2026a.

Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. In *Proc. Int. Conf. Mach. Learn.*, 2026b.

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. In *Adv. Neural Inf. Process. Syst.*, 2023.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. Libero-plus: A progressive robustness benchmark for visual-language-action models. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2026.

Yoav HaCohen, Nisan Chiprut, Benny Brazowski, Daniel Shalem, Dudu Moshe, Eitan Richardson, Eran Levin, et al. Ltx-video: Realtime video latent diffusion. *arXiv preprint arXiv:2501.00103*, 2024.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Find. Assoc. Comput. Linguist.: EMNLP 2020*, 2020.

Jinbang Huang, Yuanzhao Hu, Zhiyuan Li, Ran Qi, Yixin Xiao, Zhanguang Zhang, Mark Coates, Tongtong Cao, and Yingxue Zhang. RoboHarness: Memory-driven orchestration of heterogeneous robot policies for long-horizon planning. *arXiv preprint arXiv:2607.18060*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, et al. Openvla: An open-source vision-language-action model. In *Proc. Conf. Robot. Learn.*, 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, and Jinwei Gu. Cosmos policy: Fine-tuning video models for visuomotor control and planning. In *Proc. Int. Conf. Learn. Represent.*, 2026.

Lin Li, Qihang Zhang, Yiming Luo, Shuai Yang, Ruilin Wang, Luyao Zhang, Mingrui Yu, Zelin Gao, et al. Causal world modeling for robot control. In *Proc. Robot.: Sci. Syst.*, 2026a.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proc. Int. Conf. Mach. Learn.*, 2024b.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proc. Conf. Empir. Methods Nat. Lang. Process.*, 2024c.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. In *Adv. Neural Inf. Process. Syst.*, 2025.

Ziang Li, Dongzhou Cheng, Yibin Wang, Shiyue Wang, Xiaoyang Xu, Lingxuan Weng, Juan Wang, and Jiaqi Wang. Light-wam: Efficient world action models with state-fusion action decoding. *arXiv preprint arXiv:2606.08242*, 2026b.

Yue Liao, Pengfei Zhou, Siyuan Huang, Donglin Yang, Shengcong Chen, Yuxin Jiang, Yue Hu, Si Liu, et al. Genie envisioner: A unified world foundation platform for robotic manipulation. In *Proc. Int. Conf. Learn. Represent.*, 2026.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. In *Adv. Neural Inf. Process. Syst.*, 2023.

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

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

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

Jingwen Sun, Wenyao Zhang, Zekun Qi, Shaojie Ren, Zezhi Liu, Hanxin Zhu, Guangzhong Sun, Xin Jin, and Zhibo Chen. Vla-jepa: Enhancing vision-language-action model with latent world model. In *Proc. Eur. Conf. Comput. Vis.*, 2026.

Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

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, et al. World action models are zero-shot policies. In *ICLR 2026 the 2nd Workshop on World Models: Understanding, Modelling and Scaling*, 2026b.

Haoqi Yuan, Zhixuan Liang, Anzhe Chen, Ye Wang, Haoyang Li, Pei Lin, Yiyang Huang, Zixing Lei, Tong Zhang, Jiazhao Zhang, et al. Qwen-robotmanip technical report: Alignment unlocks scale for robotic manipulation foundation models. *arXiv preprint arXiv:2606.17846*, 2026a.

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*, 2026b.

Wenyao Zhang, Hongsi Liu, Zekun Qi, Yunnan Wang, XinQiang Yu, Jiazhao Zhang, Runpei Dong, Jiawei He, He Wang, Zhizheng Zhang, Li Yi, Wenjun Zeng, and Xin Jin. Dreamvla: A vision-language-action model dreamed with comprehensive world knowledge. In *Adv. Neural Inf. Process. Syst.*, 2025.

Zhanguang Zhang, Zhiyuan Li, Behnam Rahmati, Rui Heng Yang, Yintao Ma, Amir Rasouli, Sajjad Pakdamansavoji, Yangzheng Wu, Lingfeng Zhang, Tongtong Cao, et al. Do world action models generalize better than vlas? a robustness study. *arXiv preprint arXiv:2603.22078*, 2026.

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, et al. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models. In *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2025.

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

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, et al. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *Proc. Int. Conf. Learn. Represent.*, 2026.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Proc. Conf. Robot. Learn.*, 2023.

A Related Work

Vision-Language-Action Models (VLAs). Collecting diverse robot demonstrations is costly, motivating the transfer of knowledge from large-scale pretraining. VLAs adapt pretrained vision-language representations for control (Zitkovich et al., 2023; Kim et al., 2025; Black et al., 2025b;a; Li et al., 2024a; Bjorck et al., 2025; Zheng et al., 2026; Yuan et al., 2026a). A VLA typically builds on a large language model (LLM) backbone with a vision adapter and, in some architectures, an action expert (Black et al., 2025b;a). Thus, VLAs typically need to bridge language-oriented representations to continuous robot control. To equip VLAs with stronger visual perception and visual foresight, subsequent work augments these policies with future-image, visual-subgoal, or latent-state prediction (Cen et al., 2025; Zhao et al., 2025; Cai et al., 2026b; Zhang et al., 2025; Sun et al., 2026).

World Action Models (WAMs). WAMs build on pretrained video world models to support the spatial reasoning and dynamics understanding needed for robot control (Du et al., 2023; Bi et al., 2026; Li et al., 2026a; Ye et al., 2026b). Unlike VLAs, video world models learn representations grounded in physical dynamics, providing a natural basis for action prediction.

However, the use of large video backbones and iterative video generation makes WAM inference computationally expensive, limiting its applicability to real-time robot control. Thus, recent work are moving from shared-backbone designs toward Mixture-of-Transformers (MoT) architectures, which decouple video and action generation. Shared-backbone designs use the full world-model backbone for both video and action denoising (Kim et al., 2026; Ye et al., 2026b; Li et al., 2026a)[2]. Although effective, these designs typically incur high inference latency. MoT architectures decouple the video and action generation by using separate video and action experts With narrower action modules and distinct generation frequencies and denoising schedules for video and action (Bi et al., 2026; Li et al., 2026a), the inference latency can be significantly improved. Fast-WAM further shows that observed-frame prefilling is sufficient to provide visual knowledge from the video backbone without explicit future-video denoising (Yuan et al., 2026b), which further reduce the inference latency.

Nonetheless, most MoT-based WAMs, including Fast-WAM, typically couple the action expert's depth with its access to representations across the video backbone. Consequently, reducing the expert's depth typically restrict its direct access to some backbone layers (Ma et al., 2026) Accessing representations from all video layers therefore requires a deep action expert that must run repeatedly during action denoising, leaving WAM inference slower than that of representative VLAs. Faster-WAM pursues a complementary architectural improvement guided by the world-model-centric principle: DoT decouples action depth from video depth while preserving access to the full video hierarchy. Faster-WAM thereby directly reduces inference latency while preserving competitive control performance.

Latent Feature and KV Aggregation. A shallow action expert also needs an efficient interface to representations across the video backbone. DiT4DiT (Ma et al., 2026) and mimic-video (Pai et al., 2025) introduce action decoders conditioned on intermediate video hidden states, with extraction layers selected empirically. In LLM speculative decoding, the EAGLE family reuses target-model features for lightweight draft models, with EAGLE-3 fusing hidden states from multiple depths through a dense projection (Li et al., 2024b;c; 2025). DFlash (Chen et al., 2026a) similarly fuses selected hidden states and projects them into a lightweight diffusion drafter as keys and values. Our Lite KV-Fusion instead learns head-specific mixtures of the video backbone's native keys and values across all layers, avoiding manual layer selection and additional dense projections of backbone features.

[2](Li et al., 2026a) introduces both shared-backbone and MoT variants, but its publicly released code primarily implements the shared-backbone variant.

B THEORETICAL SUPPORT

B.1 NORM-BOUND PRESERVATION UNDER LITE KV-FUSION

Lite KV-Fusion is a standalone aggregation operator whose mixing weights depend on the output layer and head. Fix one instance of this operator, a token position p, and a head h. Let $\widetilde{k}_{\ell h}(p) \in \mathbb{R}^d$ denote the cached key from source layer ℓ, after native key normalization and RoPE, when applicable. The operator produces

$$\widehat{k}_h(p) = \sum_{\ell=1}^{L_v} \beta_{h\ell} \widetilde{k}_{\ell h}(p), \qquad \beta_{h\ell} \geq 0, \qquad \sum_{\ell=1}^{L_v} \beta_{h\ell} = 1,$$

where the last two properties follow from softmax normalization over source layers.

Proof of Prop. 2.1. The triangle inequality and nonnegative mixing weights give

$$\begin{aligned} \|\widehat{k}_h(p)\|_2 &\leq \sum_{\ell=1}^{L_v} \beta_{h\ell} \|\widetilde{k}_{\ell h}(p)\|_2 \\ &\leq \max_{\ell} \|\widetilde{k}_{\ell h}(p)\|_2 \sum_{\ell=1}^{L_v} \beta_{h\ell} \\ &= \max_{\ell} \|\widetilde{k}_{\ell h}(p)\|_2. \end{aligned} \tag{4}$$

Thus, the fusion operator preserves any common upper bound on its input key norms within a head. □

This result retains scale control on attention keys introduced by QK-Norm. Note that this does not guarantee the exact equal key norms.

B.2 ROPE PRESERVATION UNDER LITE KV-FUSION

Lite KV-Fusion aggregates keys at the same token position across video layers. Because these keys share a rotary transform, fusion preserves that transform exactly.

Fix an output layer and attention head h, and let $k_{\ell h}(p) \in \mathbb{R}^d$ be the key at position p in video layer ℓ, after key normalization but before RoPE. Here, p denotes a temporal or spatiotemporal coordinate. All video layers use the same rotary matrix $R_h(p)$, with the same frequencies, axis partition, and channel pairing. Their cached keys are therefore $\widetilde{k}_{\ell h}(p) = R_h(p) k_{\ell h}(p)$. The softmax layer weights $\beta_{h\ell}$ are independent of token position and shared across all channels within a head. Softmax acts only on the layer logits, so fusion remains linear in the keys. Define

$$\begin{aligned} \bar{k}_h(p) &= \sum_{\ell=1}^{L_v} \beta_{h\ell} k_{\ell h}(p), \\ \widehat{k}_h(p) &= \sum_{\ell=1}^{L_v} \beta_{h\ell} \widetilde{k}_{\ell h}(p). \end{aligned} \tag{5}$$

Prop. 2.2 states that the fused video key retains the shared rotary transform:

$$\widehat{k}_h(p) = R_h(p) \bar{k}_h(p). \tag{6}$$

Proof of Prop. 2.2. Each $\beta_{h\ell}$ is a scalar shared across the channels of head h, and $R_h(p)$ is identical across video layers at position p. Therefore,

$$\begin{aligned}\widehat{k}_h(p) &= \sum_{\ell=1}^{L_v} \beta_{h\ell} \widetilde{k}_{\ell h}(p) \\ &= \sum_{\ell=1}^{L_v} \beta_{h\ell} R_h(p) k_{\ell h}(p) \\ &= R_h(p) \sum_{\ell=1}^{L_v} \beta_{h\ell} k_{\ell h}(p) = R_h(p) \bar{k}_h(p).\end{aligned}$$

Thus, fusion preserves the source video keys' rotary transform, including its axes, frequencies, and channel pairing. This proves Eq. (6) for both 1D and 3D RoPE. □

This identity permits exact de-rotation after fusion, as detailed in Appx. B.4.

B.3 Misalignment between 1D Temporal and 3D Video RoPE

Consider one attention head with $d/2$ rotary pairs, and let $q_j, k_j \in \mathbb{R}^2$ denote the query and key in pair j, after any normalization. For a two-dimensional rotation $R(\theta)$, RoPE satisfies (Su et al., 2024)

$$(R(\theta)q_j)^\top R(\theta')k_j = q_j^\top R(\theta' - \theta)k_j. \tag{7}$$

The usual relative-position interpretation requires matching coordinate axes and frequencies in corresponding query and key pairs.

For the native frequency schedules, 1D action RoPE assigns all $d/2$ pairs to time, with $\nu_j = \theta^{-2j/d}$, $j = 0, \ldots, d/2 - 1$, for base $\theta > 1$. The video backbone partitions the head into even dimensions $d_t + d_y + d_x = d$, using $\omega_r^b = \theta^{-2r/d_b}$ for axis $b \in \{t, y, x\}$. Thus, video RoPE has only $d_t/2$ temporal frequencies, with a different frequency grid even when both encodings use the same base θ. For $d = 128$, the video partition is $(d_t, d_y, d_x) = (44, 42, 42)$: action RoPE uses 64 temporal frequencies, compared with 22 for video time and 21 for each spatial axis. Both encodings still use 64 rotary pairs in total.

Proof of Prop. 2.3. Let i denote the action index, $u = (t, y, x)$ the video coordinates, and $b(j) \in \{t, y, x\}$ the video axis assigned to pair j. For positive action and video frequencies ν_j, ω_j, respectively, Eq. (7) gives the unscaled attention logit

$$\mathcal{S}(i, u) = \sum_{j=0}^{d/2-1} q_j^\top R\big(\omega_j u_{b(j)} - \nu_j i\big)\, k_j. \tag{8}$$

For a spatial pair with coordinate $z \in \{y, x\}$, its relative phase can be written as

$$\omega_j z - \nu_j i = \omega_j\big(z - \rho_j(i)\big), \qquad \rho_j(i) = \frac{\nu_j}{\omega_j} i. \tag{9}$$

Thus, the action index acts as an implicit spatial reference. Advancing it by δ shifts this unwrapped reference by $(\nu_j/\omega_j)\delta$, even when the observed frame ($t = 0$) and its spatial content are fixed. This artificial displacement is determined by token order and frequency ratios, independently of scene geometry.

For a temporal pair, the relative phase is

$$\omega_j t - \nu_j i = \omega_j(t - i) + (\omega_j - \nu_j)i. \tag{10}$$

When $\omega_j \neq \nu_j$, this phase depends on the absolute action index as well as the time difference. Shifting both t and i by δ changes the phase by $(\omega_j - \nu_j)\delta$, although $t - i$ is unchanged. With the native schedules above, $\omega_j = \theta^{-2j/d_t}$ and $\nu_j = \theta^{-2j/d}$ on the temporal block, so their frequencies differ for $j > 0$. Together, the axis assignment and frequency-grid mismatch prevent a consistent relative-position interpretation. □

B.4 RETRACTED 1D ROPE FOR VIDEO TOKENS

We align fused video keys with the action module's 1D RoPE by removing their native 3D rotation and applying the action rotary basis.

De-rotation. RoPE rotates each pair of key channels by an angle determined by the token position. De-rotation applies the opposite angle to the same pair. For a video token at $p = (t, y, x)$, pair r uses $\theta_r(p) = \omega_r p_{b(r)}$, where $b(r) \in \{t, y, x\}$ selects its axis and ω_r is its native RoPE frequency. Let $\widehat{k}_{h,r}(p) \in \mathbb{R}^2$ denote this pair in the fused key of head h. The de-rotated pair is

$$\begin{aligned} \bar{k}_{h,r}(p) &= R(-\theta_r(p))\widehat{k}_{h,r}(p) \\ &= \begin{pmatrix} \cos\theta_r(p) & \sin\theta_r(p) \\ -\sin\theta_r(p) & \cos\theta_r(p) \end{pmatrix} \widehat{k}_{h,r}(p). \end{aligned} \tag{11}$$

Since $R(-\theta) = R(\theta)^\top = R(\theta)^{-1}$, applying this operation to every pair and using Eq. (6) gives

$$R_h(p)^\top \widehat{k}_h(p) = \bar{k}_h(p). \tag{12}$$

Thus, de-rotating the fused key is equivalent to de-rotating each source key before fusion. In complex notation, the same operation multiplies each pair by $e^{-\mathrm{i}\theta_r(p)}$, the conjugate of its original rotary factor.

Reapplying 1D RoPE. Let R_h^a denote the action module's 1D rotary transform, with its temporal axis, frequency grid, and channel pairing. Expressing the video token's time t_p relative to the same temporal origin and in the same units as the action indices, we obtain

$$\begin{aligned} k_h^{v\to a}(p) &= R_h^a(t_p) R_h(p)^\top \widehat{k}_h(p) \\ &= R_h^a(t_p)\bar{k}_h(p). \end{aligned} \tag{13}$$

Faster-WAM conditions only on the prefilled observed frame, for which $t_p = 0$. Since $R_h^a(0) = I$, reapplication is the identity and $k_h^{v\to a}(p) = \bar{k}_h(p)$. This operation removes the explicit video rotary factor from the keys supplied to the action module; their content can still encode spatial information. The video backbone retains its native 3D RoPE.

Note that the retracted 1D-RoPE is not limited to WAMs condition only on the prefilled observed frame. It can be directly applied to those condition on future video frames by assigning proper 1D-RoPE.

B.5 ALTERNATIVE ROPE ALIGNMENT: LIFTED 3D ROPE

Besides retracted 1D RoPE, we also attempted an alternative solution by uplift the 1D RoPE of action tokens. However, it introduces a stronger inductive bias on how action tokens attend to video tokens, which might hurt the generalizability according to our ablation study (Sec. 3.3.2).

Lifted 3D RoPE maps action timestamps into the video backbone's spatiotemporal coordinate system through spatial pseudo-anchors.

Spatial pseudo-anchors. We use the video backbone's temporal–height–width channel partition and frequency banks for action queries and keys. Action token i and video token s retain their temporal indices i and t_s, respectively. Since action tokens have no image-grid coordinates, we assign them spatial pseudo-anchors at the camera-view centers. Video keys retain their image-grid positions, so spatial rotary pairs encode offsets from these anchors. For a head dimension of $d = 128$, the temporal, height, and width blocks contain 22, 21, and 21 rotary pairs, respectively, with $\omega_r^t = 10000^{-2r/44}$ and $\omega_r^y = \omega_r^x = 10000^{-2r/42}$, indexed from zero. For a composite token grid of size $G_y \times G_x$, the normalized center $(\bar{c}_c^y, \bar{c}_c^x)$ of camera view c gives

$$c_c^y = G_y \bar{c}_c^y - \tfrac{1}{2}, \qquad c_c^x = G_x \bar{c}_c^x - \tfrac{1}{2}. \tag{14}$$

For action token i, head h, and video token s at (t_s, y_s, x_s), the phases are

$$\begin{aligned} \phi_{ihr}^{a,t} &= i\omega_r^t, & \phi_{sr}^{v,t} &= t_s\omega_r^t, \\ \phi_{ihr}^{a,y} &= c_{m_{hr}}^y \omega_r^y, & \phi_{sr}^{v,y} &= y_s\omega_r^y, \\ \phi_{ihr}^{a,x} &= c_{m_{hr}}^x \omega_r^x, & \phi_{sr}^{v,x} &= x_s\omega_r^x, \end{aligned} \tag{15}$$

Here, r ranges over the pairs of each axis, and m_{hr} selects a camera view as defined in Eq. (16). For each head h and spatial frequency rank r, the pseudo-anchor defines the coordinate lift $i \mapsto (i, c^y_{m_{hr}}, c^x_{m_{hr}})$. The relative spatial phases are now $\omega^y_r(y_s - c^y_{m_{hr}})$ and $\omega^x_r(x_s - c^x_{m_{hr}})$, independent of the action index. The spatial rotary factors cancel in action–action attention because the pseudo-anchors are shared across action tokens. These assignments apply to attention in the action module; the video backbone retains its original 3D RoPE. This design adds no learned positional parameters or additional attention operations.

Interleaving pseudo-anchors across views. Our Lifted 3D RoPE experiments on LIBERO and LIBERO-Plus use two camera views. We interleave their pseudo-anchors across spatial frequency pairs, assigning pair r in head h to view

$$m_{hr} = (r + h) \bmod 2, \tag{16}$$

where indices start at zero. The two assignments form S_2, comprising the identity and swap. With $H = 24$, each view is assigned to 12 heads at every frequency index. Height and width pairs share the same view assignment, while native frequencies and channel order are preserved.

For multi-view settings such as RoboTwin, this construction could be extended using cyclic shifts or permutations of the view indices. For C views, cyclic assignments take the form

$$m_{hr} = (r + h) \bmod C. \tag{17}$$

Alternatively, one could use $m_{hr} = \pi_h(r \bmod C)$, with π_h drawn from the permutation group S_C. Uniform replication requires $C \mid H$ for cyclic shifts or $C! \mid H$ for all permutations. For RoboTwin's three views, these choices yield three cyclic shifts or six permutations in S_3; either would assign each view to 8 of the 24 heads at every frequency index under uniform replication.

C TRAINING AND INFERENCE

Training objective. We use joint video and action flow-matching losses following Fast-WAM (Yuan et al., 2026b). Let X^v denote future video latents and X^a the demonstrated action chunk. For each modality $m \in \{v, a\}$, we independently sample standard Gaussian noise $\epsilon^m \sim \mathcal{N}(0, I)$ and a noise level $\sigma_m \in [0, 1]$ from the schedule below, constructing

$$X^m_{\sigma_m} = (1 - \sigma_m)X^m + \sigma_m \epsilon^m, \qquad u^m = \epsilon^m - X^m. \tag{18}$$

The corresponding DiT predicts the target velocity u^m. We minimize

$$\mathcal{L} = \sum_{m \in \{v,a\}} \lambda_m \, \mathbb{E}\Big[w_m(\sigma_m) \big\| f^m_\theta(X^m_{\sigma_m}, \sigma_m; c_m) - u^m \big\|^2_{\text{valid}} \Big], \tag{19}$$

where c_v contains the clean observed frame, language instruction, and proprioception, while c_a is the fused video KV context of the observed frame. Here, $\| \cdot \|^2_{\text{valid}}$ denotes the mean squared error over non-padding target elements within each sample. The observed frame remains clean and is excluded from the video prediction loss. We use $\lambda_v = \lambda_a = 1$; the attention masks are specified in Sec. 2.4.

Noise schedule and weighting. For LIBERO and RoboTwin, we independently sample $z_m \sim \mathcal{U}(0, 1)$ for each modality and set $\sigma_m = \phi(z_m)$, where $\phi(z) = 5z/(1 + 4z)$. Both modalities use the same noise-level weighting function:

$$g(\sigma) = \exp\big[-2(\sigma - \tfrac{1}{2})^2\big] - \exp(-\tfrac{1}{2}), \qquad w_v(\sigma) = w_a(\sigma) = \frac{g(\sigma)}{Z + 10^{-10}},$$

where $Z = T^{-1} \sum_{r=1}^{T} g(\phi(r/T))$ and $T = 1000$. Each DiT receives the rescaled timestep $T\sigma_m$.

Optimization. We initialize the video backbone from Wan2.2-TI2V-5B and the action expert randomly. We jointly optimize the video backbone, action expert, Lite KV-Fusion, and proprioceptive projection on downstream demonstrations. The video autoencoder and text encoder remain fixed, and text embeddings are precomputed for training. Gradients from the action loss propagate through Lite KV-Fusion into the video backbone. Optimizer settings and benchmark-specific training configurations are provided in Appx. D.1.3 and H.

Inference. For each action chunk, the video backbone performs a single prefill pass on the current observation, conditioned on the language instruction and proprioceptive state, without video denoising. Lite KV-Fusion aggregates the native KV pairs across video layers, providing a fixed conditioning context throughout action denoising. We apply retracted 1D-RoPE to the fused video keys before joint attention, as defined in Eq. (3). For LIBERO and RoboTwin, we predict 32-action chunks using $N = 10$ explicit Euler steps with $\sigma_k = \phi(1 - k/N)$, $k = 0, \ldots, N$. Starting from $X^a_{\sigma_0} \sim \mathcal{N}(0, I)$, we update

$$X^a_{\sigma_{k+1}} = X^a_{\sigma_k} + (\sigma_{k+1} - \sigma_k) f^a_\theta(X^a_{\sigma_k}, \sigma_k; c_a).$$

The final step reaches $\sigma_N = 0$. Repeated denoising is confined to the action expert; no future video frames are generated for action prediction. Benchmark-specific action-execution protocols are provided in Appx. D.2.

D EXPERIMENTAL DETAILS

D.1 IMPLEMENTATION SETTINGS

D.1.1 MODEL CONFIGURATION

Faster-WAM uses the pretrained Wan2.2-TI2V-5B video backbone and a randomly initialized, single-layer action expert. The video backbone has 30 layers, hidden width 3072, and FFN width 14336. The action expert has hidden width 1024 and FFN width 4096. Its query, key, and value projections map to an attention width of 3072, matching the video backbone's 24 heads of dimension 128; the attention output is projected back to width 1024. The action expert uses retracted 1D-RoPE (Sec. 2.3) and has no text cross-attention. We use Lite KV-Fusion as described in Sec. 2.1.

D.1.2 TRAINABLE AND FROZEN MODULES

We jointly train the Wan2.2 video backbone (5.00B parameters), action expert (28.6M parameters), and Lite KV-Fusion (720 parameters). Lite KV-Fusion learns one layer-mixing logit for each attention head, video layer, and action layer, giving $24 \times 30 \times 1 = 720$ parameters. The same mixing weights are used for keys and values, so they are counted only once. The proprioceptive projection is also trainable (36864 parameters for LIBERO and 61440 for RoboTwin). The VAE (704.69M parameters) remains frozen, and text embeddings are precomputed with the same text encoder as Fast-WAM (Yuan et al., 2026b). Appx. C describes the training objective and gradient flow.

D.1.3 OPTIMIZATION

We use AdamW to optimize all trainable modules, including the proprioceptive projection, with a peak learning rate 10^{-4}, $\beta = (0.9, 0.95)$, weight decay 0.01, and gradient clipping at 1.0. We use linear warmup for the first 5% of optimizer steps, followed by cosine decay to 10^{-6}. Video and action noise levels are sampled independently as $\sigma = 5u/(1 + 4u)$, where $u \sim \mathcal{U}(0, 1)$.

D.2 BENCHMARK PROTOCOLS

D.2.1 LIBERO TRAINING AND PREPROCESSING

We use the preprocessed LIBERO (Liu et al., 2023) dataset released by Fast-WAM (Yuan et al., 2026b), generated with MuJoCo 3.3.2. The dataset contains 1712 demonstrations with no-op actions removed, spanning 40 tasks and 277713 frames across the Spatial, Object, Goal, and Long suites. We train jointly on all four suites. Agent-view and wrist images are each resized to 224×224 and concatenated horizontally to form a 224×448 image. Actions are 7-dimensional and proprioception is 8-dimensional.

We train for 10 epochs with a global batch size of 128. Training comprises 21700 optimizer steps, including 1085 warmup steps. We evaluate the final training checkpoint.

D.2.2 LIBERO AND LIBERO-PLUS EVALUATION

Following the official LIBERO benchmark protocol, we evaluate 50 trials per task across 40 tasks in the four suites. The policy predicts 32 actions and executes 10 before replanning. We evaluate the same LIBERO-trained checkpoint on LIBERO-Plus (Fei et al., 2026) to assess generalizability under distribution shifts, using 10030 tasks across seven perturbation categories.

We evaluate with three seeds and report the mean and standard deviation of the success rate.

D.2.3 ROBOTWIN TRAINING AND PREPROCESSING

Following the multi-task setup used by Fast-WAM (Yuan et al., 2026b), Motus (Bi et al., 2026), and LingBot-VA (Li et al., 2026a), we train jointly on clean and randomized RoboTwin 2.0 data (Chen et al., 2026b). The dataset comprises 2500 demonstrations collected in clean scenes and 25000 collected in randomized scenes. The visual input consists of images from one head camera and two wrist cameras, combined into a single 384×320 image. Actions and proprioception are 14-dimensional.

We train for 5 epochs with a global batch size of 512. Training comprises 58710 optimizer steps, including 2935 warmup steps. We evaluate the final training checkpoint.

D.2.4 RoboTwin Evaluation and Action Execution

Following prior work (Yuan et al., 2026b; Bi et al., 2026; Li et al., 2026a), we evaluate on 50 RoboTwin tasks under both clean and randomized conditions, using 100 trials per task. We evaluate with three seeds and report the mean and standard deviation of the success rate. The policy predicts 32 actions and executes 24 before replanning. Actions are executed as joint-position and gripper commands, and success is determined by the environment's task-completion signal. We report average success across clean and randomized conditions.

D.3 Transfer to the LTX Video Backbone

To test whether DoT transfers to other world action models beyond Wan2.2, we implement Fast-LTX and Faster-LTX with LTX-Video-v0.9 (HaCohen et al., 2024), analogous to Fast-WAM and Faster-WAM. We train two LTX variants on LIBERO and evaluate on both LIBERO and LIBERO-Plus.

Table 5: **Transfer across video world models.** LIBERO and LIBERO-Plus success rates (%) with the sizes of the video world models and depths of the action experts. Video size counts the video DiT parameters. LTX results use one evaluation run with both environment and action-noise seeds set to 42; Faster-WAM reports mean $\pm$ std over three evaluation runs. Bold indicates the stronger result within each backbone family.

Method	Video world model	Video params.	Action depth	LIBERO Avg.	LIBERO-Plus Overall
Fast-WAM	Wan2.2-TI2V-5B	5B	30	97.6	51.5
Faster-WAM	Wan2.2-TI2V-5B	5B	1	$\mathbf{98.3 \pm 0.1}$	$\mathbf{78.3 \pm 0.1}$
Fast-LTX	LTX-Video-v0.9	1.92B	28	95.1	43.0
Faster-LTX	LTX-Video-v0.9	1.92B	1	**96.4**	**53.6**

Across both Wan2.2 and LTX-Video, DoT reduces the action expert to a single layer while improving LIBERO and LIBERO-Plus over the corresponding full-depth baseline, supporting the world-model-centric design across video backbones.

E LATENCY REPRODUCIBILITY

E.1 SCOPE AND LATENCY SOURCES

We compare the model computation required to predict one action chunk, including text and vision encoding, multimodal conditioning, and action generation. We measure π_0, $\pi_{0.5}$, X-VLA, Fast-WAM, Faster-WAM, LingBot-VA, and Motus on the same 24 GiB GPU model used to report GE-ACT's latency (Liao et al., 2026, Section 3.3). The tables report the sum of the profiled model phases, excluding external input preparation, transfers, and robot execution. Fast-WAM, LingBot-VA, and Motus exceed the GPU's memory capacity as complete pipelines, so their totals combine separately timed components. The following subsections specify the implementations, configurations, and aggregation procedure.

LIBERO and RoboTwin use separate configurations, with the action-chunk length reported alongside each latency. Among the entries in our benchmark tables, GE-ACT is the sole published latency: we retain the 200 ms reported by Genie Envisioner for a 54-action prediction with one video denoising step and five action denoising steps (Liao et al., 2026, Section 3.3), following the authors' measurement protocol.

E.2 IMPLEMENTATIONS AND MEASUREMENT PROCEDURE

E.2.1 CODE AND UPSTREAM CHECKOUTS

We use the following official implementations:

- π_0 and $\pi_{0.5}$: OpenPI, `https://github.com/Physical-Intelligence/openpi`, upstream checkout `15a9616a0094`.
- X-VLA: `https://github.com/2toinf/X-VLA`, based on upstream checkout `6bc2513f5f1c`.
- Fast-WAM: `https://github.com/yuantianyuan01/FastWAM`, upstream checkout `45d8e1458921`.
- Motus: `https://github.com/thu-ml/Motus`, upstream checkout `f771216802b8a`.
- LingBot-VA: `https://github.com/Robbyant/lingbot-va`, upstream checkout `7c6ffa9bfc4b8`.

These checkouts identify the upstream sources; we add local benchmark wrappers and timing instrumentation.

E.2.2 HARDWARE AND EXECUTION

We measure latency on a consumer GPU with 24 GiB of memory. We run one policy process at a time on a single GPU, with batch size one.

- **π_0 and $\pi_{0.5}$ (OpenPI).** JAX and jaxlib 0.5.3 with compiled inference and bfloat16 compute.
- **WAM (Fast-WAM and Faster-WAM).** PyTorch 2.7.1 and CUDA 12.8, using bfloat16 without `torch.compile`.
- **X-VLA.** PyTorch 2.1.2 and CUDA 12.1, using float32 without `torch.compile`.
- **LingBot-VA.** PyTorch 2.9.0 and CUDA 12.6, using bfloat16 model parameters without `torch.compile`.
- **Motus.** PyTorch 2.7.1 and CUDA 12.8, without `torch.compile`; the transformer components use bfloat16 with some float32 action-expert parameters, and the VAE parameters use float32.

E.2.3 TIMING PROTOCOL

We use 4,000 measured calls per component after five warmup calls. We report the arithmetic mean and standard deviation in milliseconds, with no outlier removal. For jointly measured phases, we

compute the sample standard deviation of their per-call sum; for separately measured components, we combine sample variances as described below. Standard deviations describe variation across calls within the recorded run, including occasional latency spikes; they are not confidence intervals across independent runs.

E.3 Inference Latency and Phase Breakdown

We benchmark policies using their LIBERO and RoboTwin configurations. Each measurement predicts one action chunk; Tabs. 7 and 8 report the phase timings and action-chunk lengths for the two benchmarks. The vision encoder runs for every prediction, and text encoding or embedding is included; tokenization is excluded from timing.

LIBERO. π_0, $\pi_{0.5}$, X-VLA, Fast-WAM, and Faster-WAM use ten action denoising steps. LingBot-VA uses 20 video denoising steps and 50 action denoising steps, followed by one final cache-building forward for each modality; it is measured after its observation-history buffer is full.

RoboTwin. π_0, $\pi_{0.5}$, X-VLA, Fast-WAM, and Faster-WAM use ten action denoising steps. LingBot-VA uses 25 video denoising steps and 50 action denoising steps, followed by one final cache-building forward for each modality; it is measured after its observation-history buffer is full. Motus uses ten joint video/action denoising steps and retains native video decoding. Its Qwen vision–language model runs eleven times per action chunk.

Tab. 6 defines the four inference phases used in both timing tables. Motus is included only in the RoboTwin comparison. OpenPI phases are profiled as separately compiled functions with output synchronization. X-VLA uses CUDA events around the named phases and synchronizes each profiled call; the WAM component measurements use synchronized wall-clock timing. Profiling boundaries and separate compilation can make these totals differ from native model-call times.

For Faster-WAM, the video backbone is prefilled once per observation and the fused KV cache is reused across the ten action denoising steps. Both WAM implementations prepare the action attention mask once per prediction and reuse it across the denoising steps. Instruction encoding is included in every reported prediction.

Phase	Operations by model
Text encoding / embedding	OpenPI and X-VLA: token embedding. WAM: UMT5-XXL encoding. LingBot-VA: positive/negative UMT5-XXL encoding. Motus: UMT5-XXL encoding and Qwen token embedding.
Vision encoder	OpenPI: SigLIP. X-VLA: Florence-2 vision encoder and projection. WAM and LingBot-VA: VAE encoding. Motus: VAE encoding and Qwen vision encoding.
Multimodal conditioning	OpenPI: PaliGemma image–text prefix. X-VLA: Florence-2 encoding of text and first view. WAM: robot-state encoding and video prefill. LingBot-VA: history-cache updates and video generation. Motus: Qwen multimodal encoding and T5 projection.
Action generation	OpenPI and WAM: action denoising. X-VLA: action denoising, additional-view fusion, setup, postprocessing, and instrumentation. LingBot-VA: action denoising, final cache update, and remaining model work. Motus: joint video/action denoising, video decoding, and remaining model work.

Table 6: Operations in each phase. OpenPI denotes π_0 and $\pi_{0.5}$; WAM denotes Fast-WAM and Faster-WAM.

Table 7: LIBERO latency by phase. Values are mean $\pm$ standard deviation in milliseconds.

Phase	π_0	$\pi_{0.5}$	X-VLA	LingBot-VA	Fast-WAM	Faster-WAM
Text encoding / embedding	0.205 ± 0.043	0.207 ± 0.045	0.016 ± 0.002	105.52 ± 0.12	17.48 ± 0.03	17.48 ± 0.03
Vision encoder	12.02 ± 0.09	12.01 ± 0.08	14.78 ± 0.06	68.31 ± 0.31	8.05 ± 0.04	8.09 ± 0.03
Multimodal conditioning	31.76 ± 0.09	35.08 ± 0.07	2.66 ± 0.03	1562.62 ± 2.57	20.38 ± 0.05	20.63 ± 0.06
Action generation	20.98 ± 0.09	20.42 ± 0.07	76.67 ± 0.28	3020.43 ± 5.12	150.12 ± 1.65	5.84 ± 0.10
Profiled phase total	64.96 ± 0.17	67.72 ± 0.14	94.13 ± 0.31	4756.87 ± 7.03	196.03 ± 1.67	52.04 ± 0.18
Action-chunk length	50	10	30	16	32	32

Table 8: RoboTwin latency by phase. Values are mean ± standard deviation in milliseconds.

Phase	π_0	$\pi_{0.5}$	X-VLA	Motus	LingBot-VA	Fast-WAM	Faster-WAM
Text encoding / embedding	0.215 ± 0.034	0.216 ± 0.033	0.015 ± 0.002	54.16 ± 0.06	105.44 ± 0.21	17.52 ± 0.04	17.52 ± 0.04
Vision encoder	12.01 ± 0.08	11.95 ± 0.07	20.71 ± 0.05	100.69 ± 0.68	111.82 ± 0.47	8.92 ± 0.04	8.94 ± 0.04
Multimodal conditioning	31.75 ± 0.11	35.11 ± 0.07	2.56 ± 0.02	115.06 ± 2.93	3061.35 ± 12.56	20.79 ± 0.05	20.97 ± 0.15
Action generation	20.72 ± 0.10	23.95 ± 0.08	76.62 ± 0.80	1110.35 ± 2.04	4107.50 ± 7.11	149.55 ± 1.34	5.78 ± 0.12
Profiled phase total	64.70 ± 0.21	71.23 ± 0.14	99.91 ± 0.83	1380.27 ± 4.24	7386.11 ± 15.76	196.77 ± 1.36	53.20 ± 0.26
Action-chunk length	50	50	30	16	32	32	32

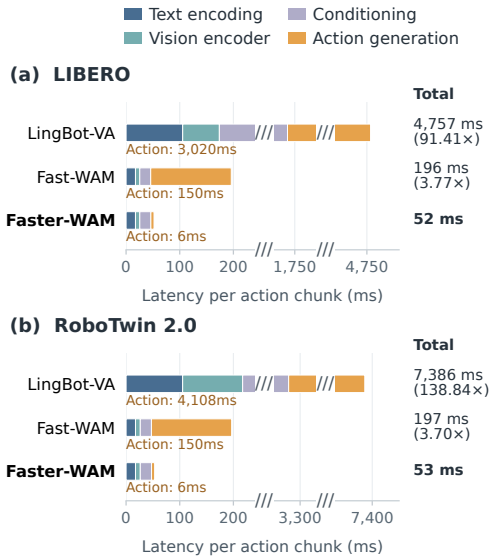


Figure 5: **Inference latency by phase.** "Action" labels show action-generation time; right-hand labels show total latency per action chunk, with ratios relative to Faster-WAM in parentheses.

The Fast-WAM and LingBot-VA totals in both tables, and the Motus total for RoboTwin, combine separately measured components. For LingBot-VA, we sum the text-encoding, VAE-encoding, and video/action-core times. For Motus, we sum the UMT5-XXL encoding time and the profiled phases of its full prediction model. These component estimates exclude module loading and transfers between CPU and GPU. For phases measured together, we sum their times within each call and compute the mean and sample standard deviation of these per-call totals, preserving covariance between phases. For separately measured components, we sum their means and combine their standard deviations as $s_{\text{total}} = \sqrt{\sum_i s_i^2}$, assuming zero covariance between components.

E.4 Inference Memory

We measure peak GPU tensor memory using the RoboTwin configurations, with all policy components resident, including the text encoder. The measurement includes model parameters, caches, and working tensors. For PyTorch, we discard five warmup calls and report the maximum over ten inference calls, synchronizing the GPU and resetting the peak counter before each call. For π_0 and $\pi_{0.5}$, JAX reports a process-wide peak that also includes loading, compilation, and warmup. Tab. 9 summarizes the results. We report memory in GiB, where 1 GiB equals 2^{30} bytes.

Policy	Peak memory (GiB)
π_0	6.43[†]
$\pi_{0.5}$	6.65[†]
X-VLA	3.46
Fast-WAM	Does not fit
LingBot-VA	Does not fit
Motus	Does not fit
Faster-WAM	21.44

Table 9: Peak GPU tensor memory for RoboTwin with all policy components resident on the 24 GiB device.

[†]JAX process-wide peak, including loading, compilation, and warmup.

Fast-WAM runs out of memory while loading its action expert with the text encoder already resident. LingBot-VA's model and required history cache together need at least 29.41 GiB, exceeding device capacity before working tensors are allocated. Motus's registered model storage alone requires 28.16 GiB. We therefore measure these models in separate components for the latency results above; no full-policy inference-memory peak is available for these three models.

F Real-Robot Experimental Setup

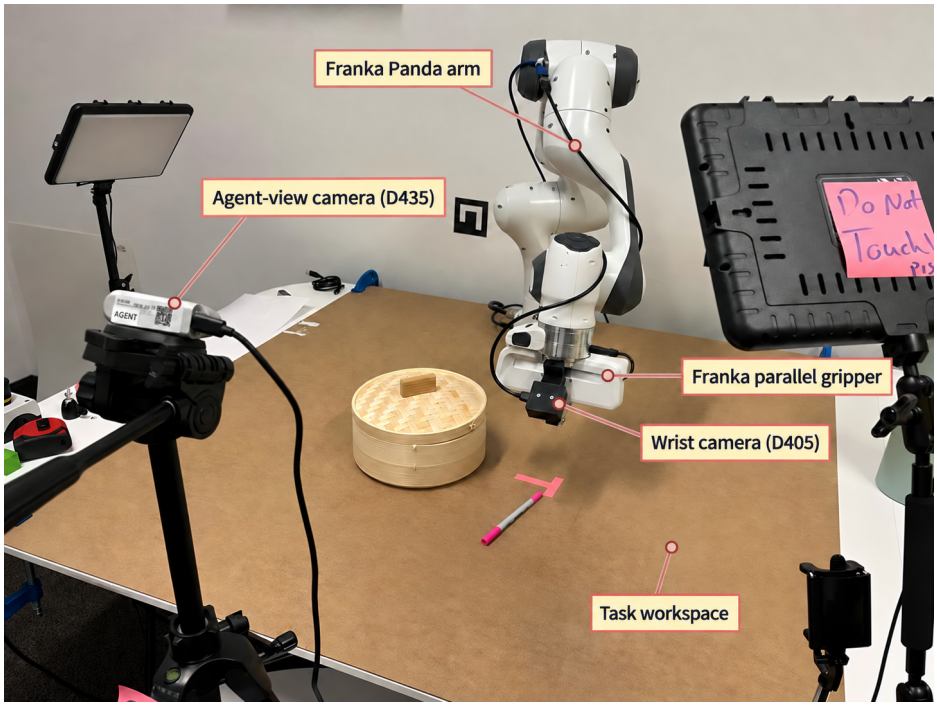


Figure 6: Real-robot experimental setup, shown with the fruit-picking task. Callouts identify the Franka Panda arm, parallel gripper, wrist-mounted D405, fixed agent-view D435, and tabletop workspace.

F.1 Hardware and Control

Fig. 6 shows our Franka Panda arm with a Franka parallel gripper mounted at a tabletop workspace, facing a fixed agent-view camera. Two Intel RealSense cameras provide complementary views of the scene: a wrist-mounted D405 observes the region near the gripper, while a fixed D435 captures the robot and task workspace. The D405 provides an RGB field of view of $87° \times 58°$ (horizontal $\times$ vertical).[3] The D435 provides an RGB field of view of $69° \times 42°$.[4] These specifications describe the camera views before image preprocessing.

The robot and cameras connect to a workstation running Linux with a real-time kernel. Each policy receives RGB images from both cameras, robot proprioception, and a task-specific text prompt. At each observation update, we read the currently available camera images and robot state. The policy produces end-effector delta commands expressed in the tool frame, together with a gripper-state command. Actions are predicted in chunks of 32, of which the first eight are executed before replanning from a new observation. Polymetis[5] provides the control interface through which these commands are sent to the robot via the Franka Control Interface.

F.2 Task Environments

The four tasks use tabletop objects arranged to test different manipulation requirements. In `place_cube`, a red wooden cube and an IKEA bowl provide a basic pick-and-place setting. The `pick_fruit` task extends this setting to sequential manipulation: fruit objects are placed inside a

[3] Manufacturer specifications: `https://www.realsenseai.com/product-family/d405-series/`.

[4] Manufacturer specifications: `https://www.realsenseai.com/products/stereo-depth-camera-d435/`.

[5] `https://facebookresearch.github.io/fairo/polymetis/`.

lidded bamboo steamer, and the robot must lift the lid, place it aside, and pick the initially concealed plum. For `push_T`, two wooden blocks arranged in a T shape provide a contact-sensitive pushing task. Finally, `dish_utensil` uses standard IKEA dishes and utensils to test organization over a longer sequence of placements, with success determined by the final arrangement.

F.3 Demonstration Collection

We collect demonstrations with a SpaceMouse, which allows the operator to control the end effector in six degrees of freedom through the same Polymetis interface used for policy execution. During teleoperation, camera observations, robot state, end-sepose, and gripper state are recorded at 10 Hz. The resulting dataset contains 100 demonstrations per task, for a total of 400 demonstrations.

To introduce variation around each nominal task configuration, we change object placements between demonstrations. For `place_cube`, the cube's initial position varies within a $10\,\mathrm{cm} \times 10\,\mathrm{cm}$ region; for `pick_fruit`, the plum is placed at different locations inside the container. We apply smaller placement perturbations to `push_T` and `dish_utensil`, including variations in utensil arrangement, because these tasks already impose greater demands on contact control and sequential execution.

F.4 Training and Evaluation

Using the combined demonstration dataset, we train policies jointly across the four tasks with language conditioning and follow each model's recommended training recipe. For Faster-WAM and Fast-WAM, this entails 30 training epochs. We evaluate each policy over 20 rollouts per task and report the fraction of successful rollouts.

For `dish_utensil`, we report a single task success rate based on its final checkpoint, which requires the complete dish-and-utensil arrangement to be correct. The rate is the number of successful rollouts divided by 20, or by 10 for LingBot-VA. Tab. 4 summarizes the results, while Appx. G provides timestamped examples of physical execution.

G Real-World Demo Rollout (Speed Comparison)

G.1 Rollout Setup

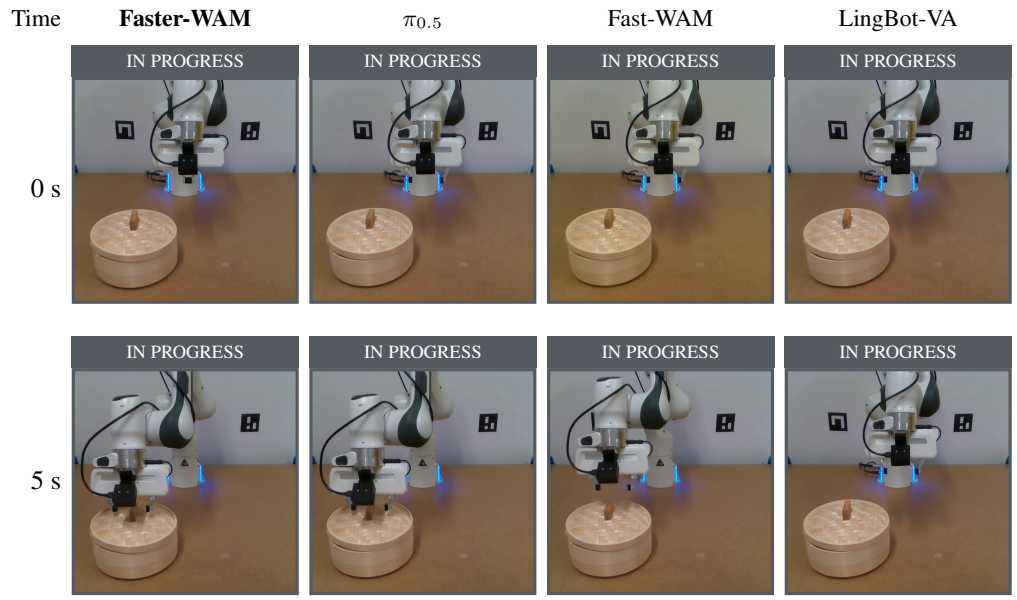
We compare four real-robot rollouts of the `pick_fruit` task, with the prompt *"Lift the lid, put it aside, and pick the black plum."* Columns identify the policies; rows sample elapsed wall-clock time from each recording's start, including pauses between movements. We show 5-second intervals through 40 seconds. Later snapshots use 20-second intervals while waiting for LingBot-VA. Tab. 10 reports the completion time of each rollout.

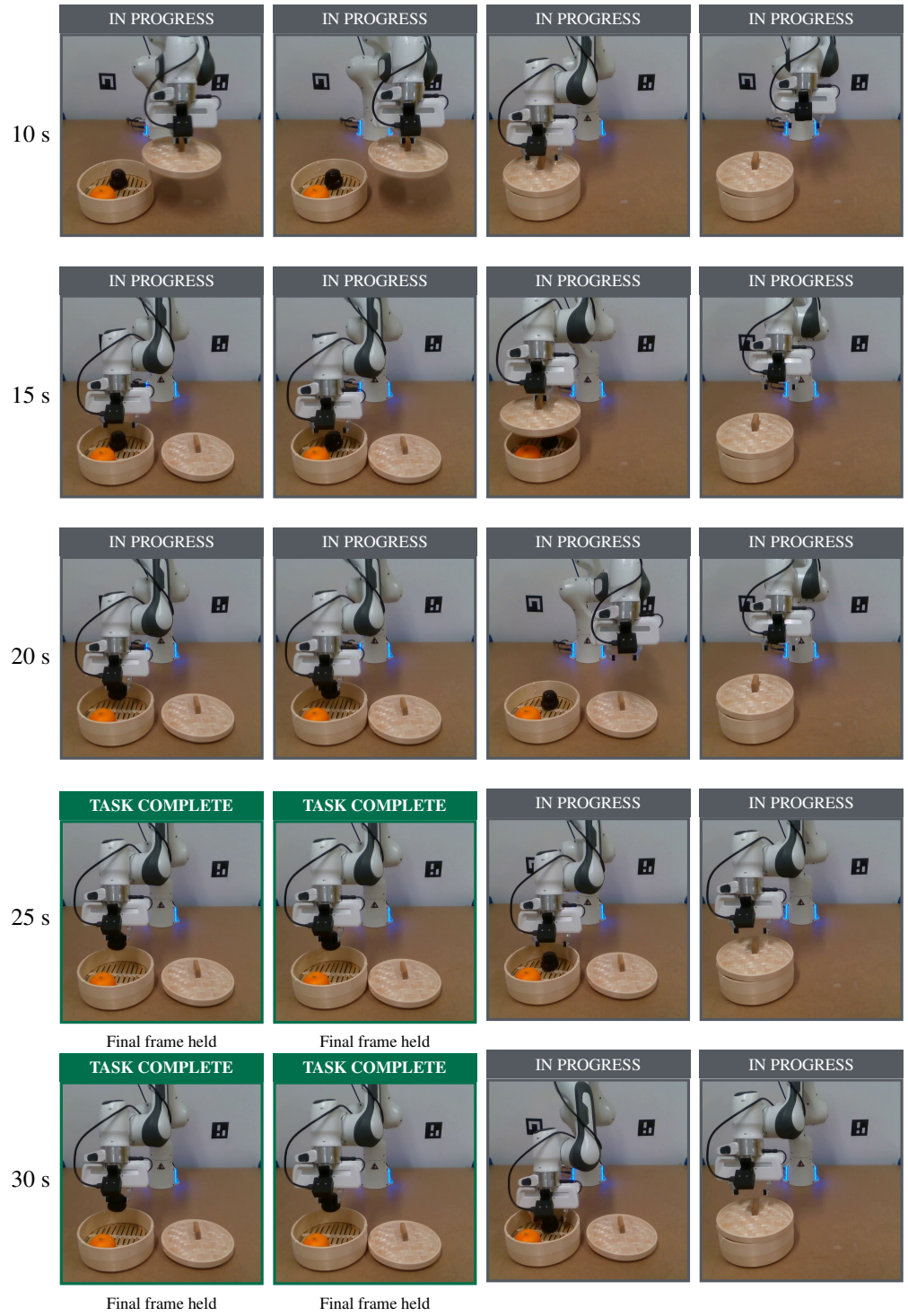
Table 10: Completion time of the `pick_fruit` rollouts in Fig. 7, measured as elapsed wall-clock time from the start of each recording.

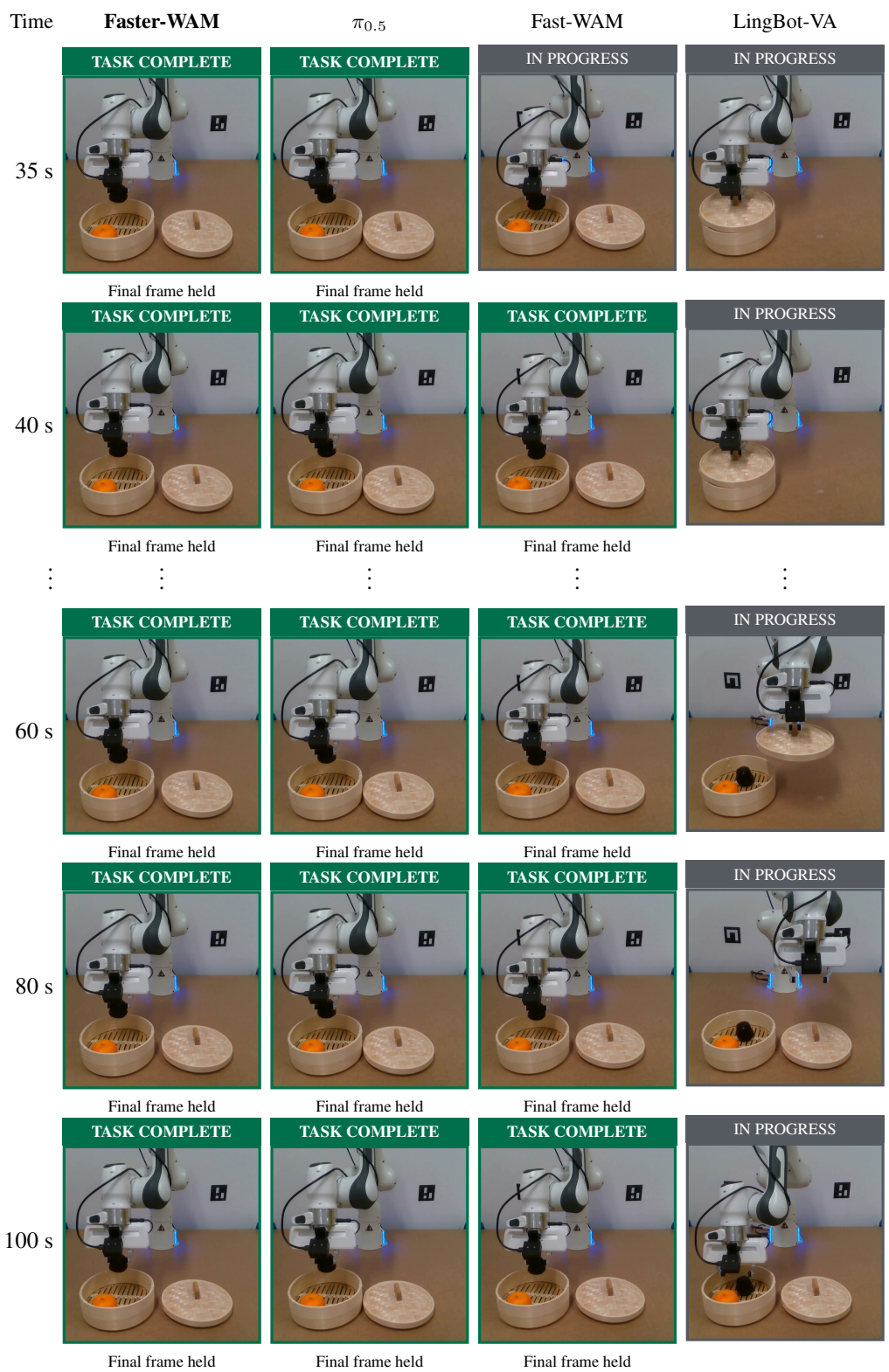
Policy	Completion time (s)
Faster-WAM	21.2
$\pi_{0.5}$	21.6
Fast-WAM	38.5
LingBot-VA	139.9

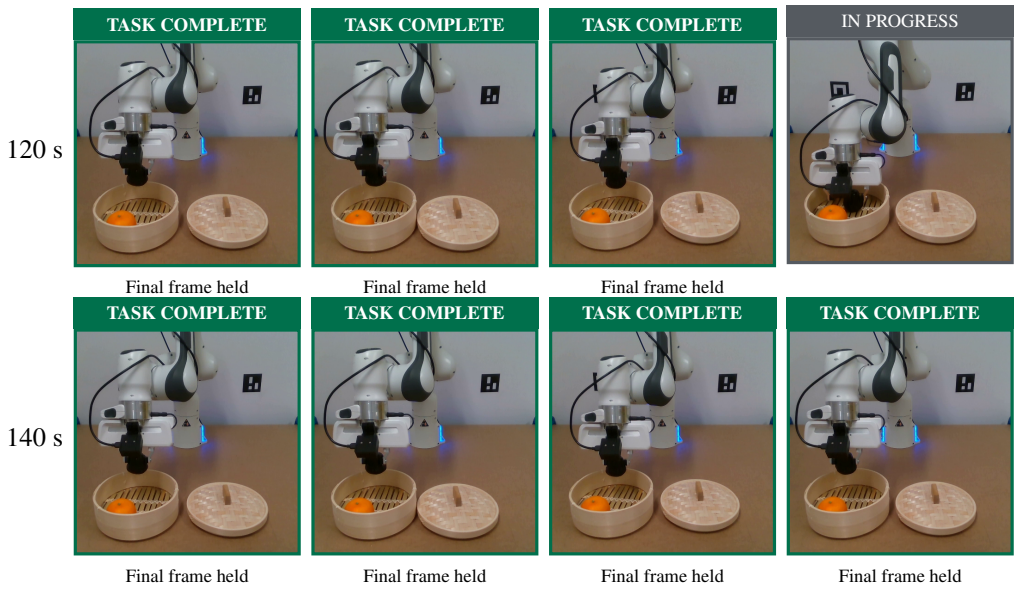
G.2 Rollout Comparison

Figure 7: **Real-world rollout speed comparison.** Snapshots at 0, 5, 10, 15, 20, 25, 30, 35, and 40 seconds, then every 20 seconds to 140 seconds. Faster-WAM and $\pi_{0.5}$ have both lifted the plum by the 25-second snapshot. The sampling interval increases after Faster-WAM, $\pi_{0.5}$, and Fast-WAM have finished. Green **TASK COMPLETE** banners and borders mark completed rollouts. The final recorded frame is held in later snapshots; gray banners indicate ongoing execution. These are illustrative rollouts, not estimates of mean task-completion time.




Time
Faster-WAM
$\pi_{0.5}$
Fast-WAM
LingBot-VA
10 s
15 s
20 s
25 s
30 s
IN PROGRESS
TASK COMPLETE
Final frame held
Final frame held
Final frame held
Final frame held


Time
Faster-WAM
$\pi_{0.5}$
Fast-WAM
LingBot-VA
35 s
40 s
60 s
80 s
100 s
TASK COMPLETE
IN PROGRESS
Final frame held


Time
Faster-WAM
$\pi_{0.5}$
Fast-WAM
LingBot-VA
120 s
TASK COMPLETE
TASK COMPLETE
TASK COMPLETE
IN PROGRESS
Final frame held
Final frame held
Final frame held
140 s
TASK COMPLETE
TASK COMPLETE
TASK COMPLETE
TASK COMPLETE
Final frame held
Final frame held
Final frame held
Final frame held

H Full Reproducibility Settings

Tables 12 and 13 summarize the Faster-WAM training recipe for LIBERO and RoboTwin. Image sizes are given as height $\times$ width.

H.1 LIBERO

Table 12: Faster-WAM training settings for LIBERO.

Setting	Value
Training	
Epochs / optimization steps	10 / 21,700
Global batch size	128
Batch per GPU / gradient accumulation	8 / 1
Optimizer / betas	AdamW / $(0.9, 0.95)$
Peak / final learning rate	10^{-4} / 10^{-6}
Schedule / linear warmup	Cosine / 5% of training steps
Weight decay / gradient clipping	0.01 / 1.0
Precision / random seed	BF16 / 42
Video backbone	Jointly fine-tuned
Data and preprocessing	
Training data	LIBERO Spatial, Object, Goal, Long; no-op actions removed
Camera views	External and wrist
Final image size per camera	External: 224×224; wrist: 224×224
Model input image size	224×448
Sequence length / sampling stride	33 / 1
Action-to-video frequency ratio	4
Action / proprioception dimension	7 / 8
Action and state normalization	Min–max
Validation fraction	0
Maximum text length	128 tokens
Model and flow matching	
Backbone model	`Wan-AI/Wan2.2-TI2V-5B`
Tokenizer model	`Wan-AI/Wan2.1-T2V-1.3B`
Video–action axis alignment	Retracted 1D-RoPE
KV fusion	Lite KV-Fusion
Video layers / width / FFN width	30 / 3,072 / 14,336
Action layers / width / FFN width	1 / 1,024 / 4,096
Heads / head dimension (both streams)	24 / 128
Video patch / input and output channels	$(1, 2, 2)$ / 48
Video attention mask	First-frame causal
Video action conditioning / action text attention	Disabled / disabled
Training / inference shift (both streams)	5.0 / 5.0
Training timesteps (both streams)	1,000
Video / action loss weight	1.0 / 1.0

H.2 RoboTwin

Table 13: Faster-WAM training settings for RoboTwin.

Setting	Value
Training	
Epochs / optimization steps	5 / 58,710
Global batch size	512
Batch per GPU / gradient accumulation	16 / 2
Optimizer / betas	AdamW / $(0.9, 0.95)$
Peak / final learning rate	10^{-4} / 10^{-6}
Schedule / linear warmup	Cosine / 5% of training steps
Weight decay / gradient clipping	0.01 / 1.0
Precision / random seed	BF16 / 42
Video backbone	Jointly fine-tuned
Data and preprocessing	
Training data	RoboTwin 2.0
Camera views	External, left wrist, and right wrist
Final image size per camera	Head: 256×320; each wrist: 128×160
Model input image size	384×320
Sequence length / sampling stride	33 / 1
Action-to-video frequency ratio	4
Action / proprioception dimension	14 / 14
Action and state normalization	Z-score
Validation fraction	0.01
Maximum text length	128 tokens
Model and flow matching	
Backbone model	`Wan-AI/Wan2.2-TI2V-5B`
Tokenizer model	`Wan-AI/Wan2.1-T2V-1.3B`
Video–action axis alignment	Retracted 1D-RoPE
KV fusion	Lite KV-Fusion
Video layers / width / FFN width	30 / 3,072 / 14,336
Action layers / width / FFN width	1 / 1,024 / 4,096
Heads / head dimension (both streams)	24 / 128
Video patch / input and output channels	$(1, 2, 2)$ / 48
Video attention mask	First-frame causal
Video action conditioning / action text attention	Disabled / disabled
Training / inference shift (both streams)	5.0 / 5.0
Training timesteps (both streams)	1,000
Video / action loss weight	1.0 / 1.0