SPLINEWAM: ADAPTIVE ACTION HORIZONS FOR WORLD ACTION MODELS VIA B-SPLINE REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

World action models (WAMs) are large embodied policies that jointly predict future video and the actions to execute, emitting a fixed-length action chunk per inference call. Such a policy allocates its computational budget uniformly in time, unable to execute for longer over free-space motion or to spend more inference on contact-rich manipulation, which limits the throughput a WAM can reach when served in the cloud. We present SplineWAM, which adaptively compresses the action trajectory into a fixed-size window of cubic B-spline parameters, fitting the knot times to the characteristics of the motion. One parameter budget then decodes into chunks of varying temporal resolution and duration, and both the executed span and the interval until the next policy call follow from the prediction itself. Aligning the video supervision to the fitted knot times of the demonstration rather than to a uniform grid concentrates the supervised frames where the action trajectory is complex. For asynchronous deployment we introduce Jacobian-Pullback Real-Time Chunking (JP-RTC), which imposes chunk continuity on the decoded raw actions the robot executes rather than on the spline parameters, and corrects the parameters through the decoder so that the executed prefix agrees with the actions already committed. On LIBERO-Plus and RoboCasa, SplineWAM improves success rate over an action chunking WAM by 8.2 and 4.4 points while cutting policy calls per episode by 22% and 26%. On three bimanual real-robot tasks under asynchronous execution, it leads or matches the baseline while decoding 1.2 to 1.6 times as much executed motion per call.

1 INTRODUCTION

World action models (WAMs) have emerged as a practical way to transfer the dynamics knowledge of large video generative models into robot manipulation policies (Ye et al., 2026b;a; Li et al., 2026; Zhang et al., 2026; Zhou et al., 2026; Yuan et al., 2026). By supervising future observations alongside future actions during training, a WAM learns not only which action to emit but also what that action does to the world. WAM designs differ widely in how the video is modelled and how tightly that model is coupled to action prediction, which has been explored thoroughly (Yuan et al., 2026; Pai et al., 2025; Ye et al., 2026b).

On the other hand, the way actions are represented and predicted has received less attention. Almost all current WAMs inherit the action chunk of Zhao et al. (2023): a fixed number of actions sampled on a uniform temporal grid. Action chunking is a strong default, and its benefits for behaviour cloning are by now well understood (Zhang et al., 2025), but it ties three quantities together that need not be tied: the number of predicted parameters, the number of executed control steps, and the temporal resolution of the prediction.

However, manipulation is not temporally uniform: reaching through free space is smooth, tolerant of error and predictable far ahead, whereas grasping, aligning and inserting need fine temporal resolution and can be committed to only briefly. A fixed chunk therefore spends more inference than free-space motion needs, and at the same time cannot replan promptly enough once contact begins. Both costs weigh heavily on an embodied model of this scale, since every policy call is a full forward pass of a video-pretrained backbone. A replanning interval too coarse for contact leaves the robot

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.

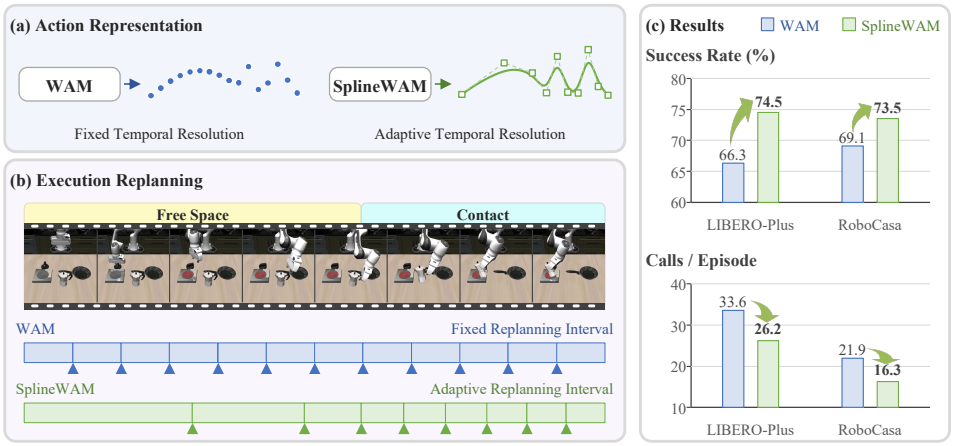


Figure 1: (a) A WAM predicts a dense, uniform chunk of discrete actions, whereas SplineWAM predicts a continuous trajectory from a few adaptively spaced knots and their control points. (b) Those knots also set the replanning schedule, so chunks stay long through free space and shorten on contact. (c) The result is fewer policy calls per episode at a higher success rate.

acting on increasingly stale observations, and redundant calls over free space consume inference for little task progress. Once such a policy is served to a fleet from shared cloud infrastructure, these redundant calls become a throughput bottleneck. A promising solution is to let the action representation carry its own horizon, so that the reach and the resolution of a prediction follow the motion rather than being fixed at design time.

To this end, we replace the action chunk of a WAM with a fixed-size window of B-spline parameters (Figure 1). A B-spline describes a smooth curve through a list of *knot times* and *control points*. The policy predicts a fixed number of such pairs, and a decoder recovers executable actions from the spline at control steps. The number of decoded actions follows the predicted knot times, so one parameter budget covers a variable execution span, and with it a variable interval until the next policy call. Adaptive fitting also places knots densely where the trajectory is hard to approximate, so the temporal resolution of the prediction follows the complexity of the motion. The knot times further define the temporal axis of the video supervision. During training we sample video targets at the fitted knot times of the demonstration rather than on a uniform grid. Both branches are then supervised on one common non-uniform axis, and future frames receive supervision most densely where the action trajectory is most complex.

Deployment on a real robot additionally requires asynchronous execution, for which real-time chunking (Black et al., 2025) offers a training-free solution with prefix guidance. Such a method is effective under action chunking, where the prefix is simply the leading entries of the generated variable, but in spline parameter space, where actions decode nonlinearly from the parameters, it is both inefficient and inflexible. We therefore propose Jacobian-pullback real-time chunking (JP-RTC), which instead forms the continuity residual on decoded actions and pulls it back through the decoder's local Jacobian.

Our contributions are threefold:

- We introduce a B-spline action representation into the action branch of a WAM, and align the video supervision to the fitted knot times of the demonstration. This fixes the parameter budget and lets the represented duration vary, so that a single prediction carries its own horizon and resolution.
- We propose JP-RTC, which imposes chunk continuity on decoded actions and pulls it back through the decoder's local Jacobian, and thereby realise asynchronous, efficient and adaptive inference that is deployable on a real robot.
- We evaluate on two simulated suites and on three bimanual real-robot tasks under asynchronous execution, ablating the representation of alignment and the fitting tolerance. SplineWAM gains 8.2 and 4.4 points of success rate over an action chunking WAM on the two suites while issuing 22% and 26% fewer policy calls per episode, and decodes $1.2 \sim 1.6$ times as much executed motion per call on the real robot.

2 Related Work

World action models and adaptive horizons. Policies that predict future observations alongside future actions have been developed in diverse forms (Wu et al., 2024; Ye et al., 2026b;a; Li et al., 2026; Zhang et al., 2026; Guo et al., 2026; Zhou et al., 2026; Pai et al., 2025; Yuan et al., 2026; Cai et al., 2026), differing in how video is modelled and coupled to action prediction, but their action branches generally predict uniformly sampled, fixed-length chunks.

Future prediction makes WAMs expensive to run, and several works therefore target the cost of action inference. Yuan et al. (2026) use an action-only test-time path, Guo et al. (2026) denoise video and action streams asynchronously, and Cai et al. (2026) decouple world prediction from action execution, adapting the horizon over which observation context is routed between the two while the action branch still emits a chunk on a fixed temporal grid. These approaches reduce the cost of producing a chunk, or vary how often it is conditioned, while retaining its fixed temporal support and sampling resolution. Outside the WAM literature, Liang et al. (2026) select chunk sizes at inference time from predicted action entropy, while Pan et al. (2026) use visual discrepancy to trigger truncation and replanning. These approaches obtain adaptivity from auxiliary signals during inference; in SplineWAM, the horizon is encoded directly by the action representation before execution begins.

Action representations. Recent work represents actions with tokens derived from frequency-domain transforms (Pertsch et al., 2025), learned tokenizers (Liu et al., 2025; Dong et al., 2026), prefix-decodable orderings (Liu et al., 2026), and quantized or hierarchical skills (Mete et al., 2024; Lee et al., 2024); another line learns latent actions from action-free video for cross-embodiment transfer (Ye et al., 2024; Bu et al., 2025; Tharwat et al., 2025; Chen et al., 2025). The temporal support of each represented segment remains fixed, while the number, hierarchy, or dimensionality of tokens varies. SplineWAM reverses this design: it fixes the parameter budget while allowing the represented duration to vary, which makes the action representation usable as an execution schedule.

B-spline actions. Zhou et al. (2025) encode fixed-length action sequences as continuous or discrete spline tokens, while Lyu et al. (2025) use splines before residual quantization. Han et al. (2026) parameterise actions by knots and control points, predict a fixed-size local window, and support time rescaling and resampling. We adopt its cubic representation and local-window construction, but use the predicted knot times to determine the execution horizon and replanning interval in an asynchronous WAM.

3 Method

3.1 Architecture

SplineWAM changes only what the action branch of a WAM predicts and leaves the rest of the model untouched, so any WAM that trains an action branch alongside a future-observation objective can adopt it. A sample consists of L_{video} observation frames, encoded to video latents, and the B-spline action window of Section 3.2 aligned to the current observation. Video latents and action parameters receive independent noise and independent flow-matching (Lipman et al., 2022) timesteps, are embedded by their respective branches, and are then processed jointly under a shared attention mask. The action loss is defined on spline parameters, not on decoded actions. Appendix D specifies the backbone, the training objective and its masking.

3.2 B-spline actions

B-spline action curves. Let a demonstration segment be a sequence of raw actions $\{(t_i, a_i)\}_{i=1}^{T}$ with $a_i \in \mathbb{R}^d$, where the time parameter t_i is the integer frame index in the source episode. We represent the segment by a cubic B-spline curve of degree $k = 3$,

$$a(u) \;=\; \sum_{j=0}^{N_c} N_{j,3}(u)\, c_j, \tag{1}$$

with basis functions $N_{j,3}$ induced by a knot vector U and control points $c_j \in \mathbb{R}^d$. All action dimensions share a single knot vector, so the temporal structure of the segment is described once rather than per dimension. We write the representation and its decoder as $z = [U, C]$ and $A = D(z)$, where C collects the control points and D evaluates Equation 1 at integer control steps to recover executable actions. The representation is smooth by construction, has local support, and can be resampled at any control frequency. Because a WAM needs a fixed output shape, the policy predicts a fixed-size local slice of the episode spline rather than a whole trajectory; Appendix D specifies how such a window is constructed and anchored to the current observation.

Fitting with adaptive knots. How the knots are placed decides whether the representation can carry a horizon: only an error-driven fit lets the duration a window spans vary with the motion, so rather than spread knots uniformly over a fixed-length chunk at a fixed compression ratio as Zhou et al. (2025) do, we follow Han et al. (2026) in fitting the whole episode trajectory under a bound on the reconstruction error, letting the number and spacing of knots follow the trajectory. Given a demonstration we fit Equation 1 by least squares with an adaptive knot insertion loop in the style of FITPACK (Dierckx, 1995): fit with the current knots, insert a knot where the residual is largest, and repeat. Actions are normalised to $[-1, 1]$ over the dataset before fitting, and insertion halts once the largest elementwise deviation in that normalised space,

$$E_{\max} = \max_{i,m} \left| \hat{a}_m(t_i) - a_m(t_i) \right|, \tag{2}$$

taken over sample indices i and action coordinates $m \in \{1, \ldots, d\}$, falls below a tolerance ε. Smooth stretches are then satisfied by few knots while high-curvature or contact-rich stretches attract many, so the knot distribution encodes local trajectory complexity and the temporal resolution it demands. The tolerance is the only knob of the representation and controls that resolution monotonically.

3.3 Training: Video–Action Temporal Alignment

Because the action window carries explicit knot times, the video supervision no longer has to live on a grid of its own. Figure 2 contrasts the possibilities. A standard WAM subsamples the video stream at a fixed stride, which is set independently of the action window and can leave its later part unsupervised. Once the window supplies knot times there are two ways to place the two supervisions on a common time axis. *Time-window alignment* keeps a uniform temporal grid over the local window, matching standard WAM practice. *Knot-grid alignment* instead walks a fixed stride along the knots rather than along time: we take the future knot times from the window at that stride, prepend the current frame, and use the resulting frame offsets to select video targets, marking repeated tail knots as padding. These are the knot times of the offline fit of Section 3.2, cached with the demonstration. Knot-grid alignment makes the video supervision share the non-uniform temporal structure of the action supervision, so that frames are supervised most densely precisely where the action trajectory is complex, and the world model is asked to predict the future at the same resolution at which the policy must act there. All three variants supervise the same number of frames at the same subsampling ratio and differ only in which frames those are; Section 4.2 measures what the choice is worth.

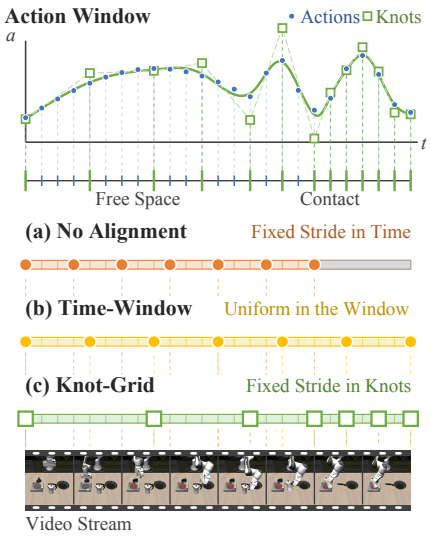


Figure 2: Three ways to sample the video stream against one action window. The time axis above marks every action step, with the predicted knots among them.

3.4 Inference

Decoding a window into actions. Decoding a predicted window at integer offsets $s = 0, 1, \ldots, \lfloor t_{\max} \rfloor$ yields $M = \lfloor t_{\max} \rfloor + 1$ actions, a count set by the predicted knot times and *not* by

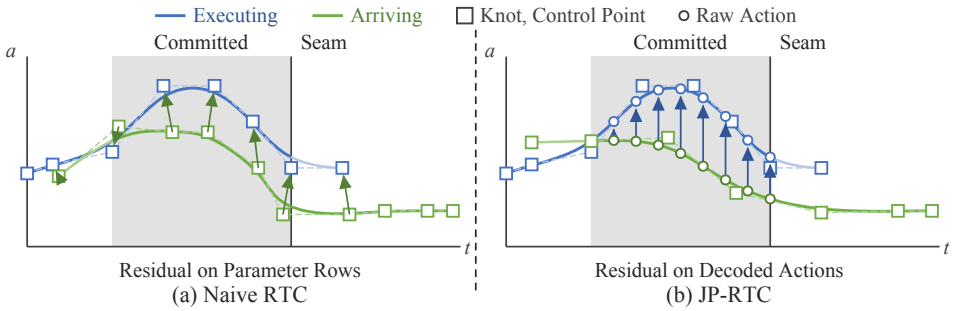


Figure 3: Where the continuity constraint is imposed; arrows show what each rule pulls together. (a) Naive RTC must align parameter rows, including those outside the committed steps whose cubic support reaches into them. (b) JP-RTC constrains the decoded actions instead, so the two windows need neither matching rows nor equal knot counts.

the number of rows: at the same L_{BSP}, widely spaced knots give a large M and dense knots a small one. The decoded length is therefore a property of the prediction rather than a hyperparameter, and it determines how long the robot can act before the policy must run again, which is what adaptive horizons require. At execution time we give the decoder a budget of $H \leq L_{\mathrm{BSP}}$ leading *parameter rows*, the robot executes every control step they decode to, and the policy is called again when those steps run out. A call therefore commits to a slice of the parameterisation rather than to a step count, and the span it covers is whatever the knot times in that slice decode to, so at a fixed H the executed span and the replanning interval vary between calls with the phase of the task.

Asynchronous execution. Under asynchronous execution the robot keeps acting while the next window is being computed, so the prefix of the new prediction that overlaps the committed actions must agree with what is already being executed. Real-time chunking (RTC) (Black et al., 2025) offers a training-free solution: it poses this as an inpainting problem and adds a pseudoinverse-guidance term to the sampler's velocity, formed from the residual between a mask-weighted prefix of the previous chunk and the one-step estimate of the chunk being generated, so that the new chunk is drawn towards agreement with the committed prefix.

For RTC the guidance target and the generated variable live in the same space, which for a per-timestep action chunk is also the space the robot executes. For a spline they are different spaces. The executed actions are $A = D(z)$ for a parameter window $z = [U, C]$, and D is nonlinear in z: one control point influences several timesteps, neighbouring control points have overlapping support, and the knot times reparameterise the curve. A fixed number of parameter rows also spans a variable amount of real time, so a prefix defined by row count need not cover the actions the robot has committed to. Masking rows of z therefore enforces parameter similarity when what asynchronous execution requires is agreement on executed actions.

Jacobian-pullback real-time chunking. We impose the constraint where it is observed and let the generated variable follow; we call the result *Jacobian-pullback real-time chunking* (JP-RTC); Figure 3 contrasts the two choices. The residual is formed on decoded actions. We decode the prefix of the arriving window over a set S of control steps and compare it against the actions the robot has committed to, namely the overlapping segment of the previous plan.

What remains is to convert a required change of actions into a change of parameters, and this is what the decoder's Jacobian J_S supplies. It is the local linear map from a perturbation of the knot times and control points to the resulting displacement of the decoded prefix, so its transpose returns a correction expressed on actions to the parameters that would produce it. In doing so it accounts for the two properties that make the spline unsuitable for direct constraint: a single control point influences several control steps, and a knot displacement retimes the curve rather than translating it. Steering the sampler along the resulting direction gives the corrected velocity

$$v_{\mathrm{JP}}(z_t^\tau, o_t, \tau) = v(z_t^\tau, o_t, \tau) + \min\left(\beta, \frac{1-\tau}{\tau \cdot r_\tau^2}\right) \delta^\star, \qquad \delta^\star = J_S^\top \left(J_S J_S^\top + \lambda I\right)^{-1} r_S, \tag{3}$$

at flow-matching time τ, where r_S is the residual on decoded actions, weighted by a mask that decays over the overlap as in RTC, J_S is evaluated at the sampler's current estimate of the clean

window, and the guidance weight is that of RTC. The bracketed $\delta^\star$ is the parameter correction, obtained from a least-squares solve for the perturbation of the window that best cancels the residual, regularised by λ. A prefix of a few control steps is typically shorter than the interval a single knot span covers, so several parameters displace the decoded actions in nearly the same manner and the solve is genuinely underdetermined. The role of λ is to select a correction within that ambiguity; Appendix E derives it, identifies it with the precision claimed for the match, sets its scale from J_S, and records the approximations the implementation makes. For a per-timestep chunk J_S reduces to a selection of rows and Equation 3 recovers the RTC update, so JP-RTC is a strict generalisation which departs from RTC only to the extent that the decoder departs from row selection.

Writing the residual after decoding also frees the two windows from having to be parameterised alike. JP-RTC never compares rows, so the arriving window is free to cover the committed steps with a different number of knots placed at different times and still be corrected to the same executed actions. What it requires is not a matching parameterisation but freedom in the rows that reach the constrained steps, and the regularisation resolves the residual ambiguity when those rows leave the solve underdetermined.

4 Experiments

We ask whether a B-spline action window improves a WAM policy and where the gains come from (Section 4.2), how much temporal compression is useful and how much of a prediction should be executed (Section 4.3), and whether the representation holds up on a real robot (Section 4.4).

4.1 Setup

We report results on two simulated suites. **LIBERO-Plus** (Fei et al., 2025) extends LIBERO (Liu et al., 2023) into a robustness analysis of vision-language-action models by perturbing seven factors: background, camera pose, language instruction, lighting, object layout, initial state and sensor characteristics. **RoboCasa** (Nasiriany et al., 2024) provides large-scale simulation of 24 distinct kitchen tasks on a Franka Panda arm.

All methods share the Fast-WAM backbone (Yuan et al., 2026). We use the default fitting tolerance of each benchmark: $\varepsilon = 0.01$ on LIBERO-Plus, and $\varepsilon = 0.08$ on RoboCasa. Tolerances are absolute values in the normalised action space of Section 3.2, chosen per suite for comparable knot density. The executed row budget of Section 3.4 is $H = 12$ on LIBERO-Plus and $H = 20$ on RoboCasa, chosen from the sweep of Section 4.3.

Baselines are **Action Chunking**, the unmodified Fast-WAM predicting a raw action chunk; **BEAST** (Zhou et al., 2025), fixed-length B-spline tokenization of a fixed-duration sequence; **Naive B-Spline**, the representation and training recipe of Han et al. (2026) without our alignment; and **Naive B-Spline w/o video**, the same variant trained with $\lambda_v = 0$ so that the auxiliary video term is removed. BEAST and Naive B-Spline denote the action representations of those papers ported into the Fast-WAM backbone, not their original models, and every baseline shares our backbone, demonstrations and training budget.

4.2 Action representation and video–action alignment

In Table 1, SplineWAM-Knot improves on the action chunking WAM by 8.2 points on LIBERO-Plus and 4.4 on RoboCasa while issuing 22% and 26% fewer policy calls per episode, and knot-grid alignment is the better of the two alignments on both suites, by 1.2 and 1.6 points.

BEAST does not benefit: it applies spline tokenization while keeping the duration of a token block fixed, and lands 0.3 points above the baseline on LIBERO-Plus while falling 4.1 points below it on RoboCasa. A spline parameterisation alone therefore does not account for the gains, since BEAST supplies one and does not obtain them; what separates it from the other spline rows is that the duration it represents is fixed in advance.

The two Naive B-Spline rows differ only in whether the auxiliary video term is present, and removing it costs 10.7 and 14.5 points. Joint video–action training is therefore doing substantial work for the action branch, which agrees with the finding of Yuan et al. (2026).

Table 1: Success rate and inference efficiency by action representation, under synchronous execution at the default tolerance of each suite. Success is episode-weighted, Calls/ep. counts policy-model invocations, and Steps/call denotes the mean control steps executed per invocation. SplineWAM-Window and SplineWAM-Knot differ only in video–action alignment.

	LIBERO-Plus ($\varepsilon = 0.01$)			RoboCasa ($\varepsilon = 0.08$)		
Action representation	Success	Calls/ep.	Steps/call	Success	Calls/ep.	Steps/call
Action Chunking	66.3	33.6	8.00	69.1	21.9	16.00
BEAST	66.6	35.5	8.00	65.0	24.6	16.00
Naive B-Spline w/o video	60.0	38.7	7.92	55.0	32.7	14.26
Naive B-Spline	70.7	27.3	8.36	69.5	16.9	17.98
SplineWAM-Window	73.3	27.1	8.39	71.9	17.5	18.10
SplineWAM-Knot	**74.5**	**26.2**	**8.41**	**73.5**	**16.3**	**18.18**

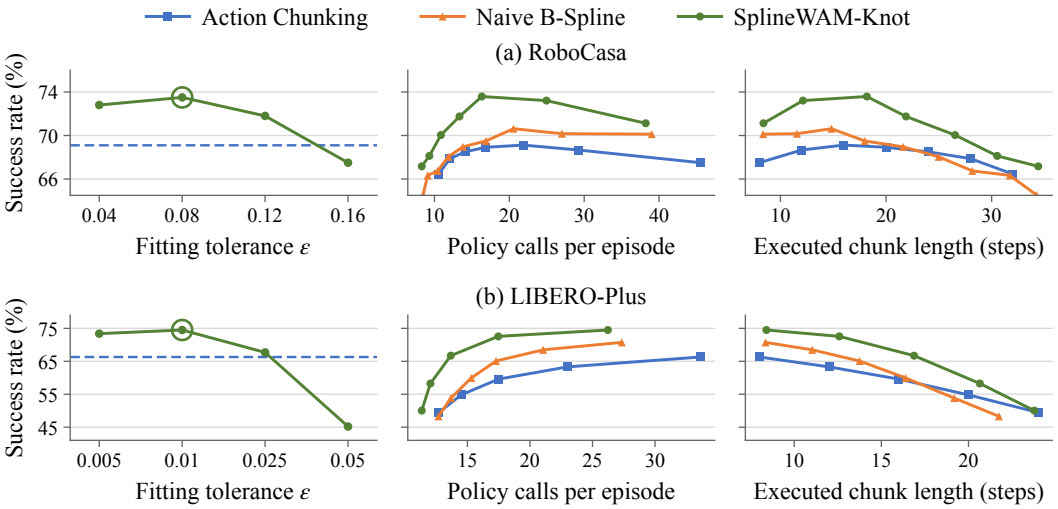


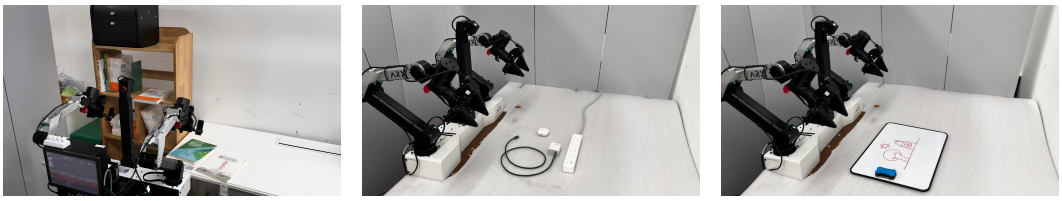
Figure 4: Ablations on (a) RoboCasa and (b) LIBERO-Plus: success rate against the fitting tolerance ε (left), and against the two costs of the executed length (middle, right). Dashed line is the action chunking baseline; the circled point is the tolerance we deploy.

4.3 Ablations

The comparison of Section 4.2 evaluates the representation at a single operating point, which is determined by two quantities. We ablate each of them in turn. The fitting tolerance ε sets how densely knots are placed in the training data, a property of the dataset the policy is fitted to; the row budget H is chosen at deployment time on a trained model.

Fitting tolerance. The left panel of each row of Figure 4 sweeps the fitting tolerance from a setting tight enough to track the demonstrations almost point by point to one loose enough to discard most knots. Each point is a separate SplineWAM-Knot model trained under the identical budget on demonstrations refitted at that tolerance, ε being a preprocessing parameter the model is never conditioned on.

The curve is flat at the tight end and collapses steeply at the loose one. Already at the tightest setting, where little temporal compression is applied, most of the improvement over action chunking is in hand, so compression is not the source of the accuracy gains; the continuous smooth parameterisation is. The loose end is unambiguous: once the knots that resolve contact-rich phases are gone, accuracy falls below the action chunking baseline and nothing downstream recovers what the representation has discarded. The sweep therefore exhibits the trade the tolerance governs, compression rate against success rate, from which we select the deployed tolerance of each suite.



(a) Arrange Bookshelf (b) Charge Earphone (c) Clean Whiteboard

Figure 5: The three real-robot tasks, each stressing a different temporal profile: (a) long-horizon mobile manipulation, (b) high-precision alignment, and (c) repetitive motion under a visual decision.

Table 2: Real-robot results over 30 trials per task and method, all under asynchronous execution. Success rate (SR) requires every scored node of Table 6; progress (PG) is the fraction reached, averaged over trials. Chunk is the mean control steps executed per policy call.

	Arrange Bookshelf			Charge Earphone			Clean Whiteboard		
Model	SR (%)	PG (%)	Chunk	SR (%)	PG (%)	Chunk	SR (%)	PG (%)	Chunk
Action Chunking	20.0	52.2	32.0	**26.7**	**73.3**	32.0	50.0	67.5	32.0
SplineWAM, Naive RTC	10.0	35.6	**39.8**	13.3	43.3	**46.5**	46.7	68.3	**53.3**
SplineWAM, JP-RTC	**23.3**	**54.4**	38.6	**26.7**	57.5	45.7	**60.0**	**74.2**	52.4

Executed length. The second ablation holds the trained model fixed and involves no retraining: it changes only the row budget H, and with it how much of a prediction is committed. We sweep H from short to long; since those rows decode to an adaptive number of steps, the executed span it yields is a mean rather than a setting. The middle and right panels of Figure 4 plot the result against both measures of what an executed length costs, the mean number of policy calls an episode requires and the mean number of control steps executed per call.

Matching the methods on inference cost gives a more direct and fairer account of what an adaptive executed length is worth, since a method that replans more often should be expected to score higher. On both suites the SplineWAM-Knot curve lies above the action chunking curve everywhere the two overlap, on either axis, and matching the *baseline's* best accuracy requires 2.2 times fewer policy calls per episode on RoboCasa and 2.5 times fewer on LIBERO-Plus; Naive B-Spline sits between the two throughout. The two suites do disagree about the shape of the trade: on RoboCasa every curve is an inverted U, so committing too little of the prediction also hurts, whereas on LIBERO-Plus all three fall monotonically and the shortest budget is always best. How much of a window to commit is therefore a deployment-time choice, tuned per suite, while the span that committed window covers remains adaptive within it.

4.4 Real-robot experiments

We build three real-robot tasks, shown in Figure 5, chosen so that the temporal structure of the demonstrations differs as much as possible between them: **Arrange Bookshelf**, a long-horizon mobile manipulation task; **Charge Earphone**, a high-precision task whose two insertions demand tight alignment at the connector; and **Clean Whiteboard**, whose wiping strokes are highly repetitive and therefore highly compressible. Every method runs asynchronously with RTC. We score each trial against a set of annotated task nodes: progress (PG) is the fraction of nodes reached and success (SR) requires all of them; Appendix B describes the tasks, the nodes and the protocol in full.

Results. SplineWAM with JP-RTC exceeds the action chunking baseline on four of the six accuracy columns of Table 2 and matches it on a fifth. The largest margin is on Clean Whiteboard, the task designed to be the most compressible, at 10.0 points of success rate and 6.7 of progress; on Arrange Bookshelf it exceeds the baseline by 3.3 and 2.2 points. The one loss is instructive: on Charge Earphone the two reach the same success rate while the baseline is ahead on progress, and the gap sits at the socket insertion in the middle of the task, itself a fine manipulation stage. A continuous representation is the less reliable of the two at an insertion the demonstrations perform identically every time, a real cost on the one task built around precision, and it costs no completed trials. Meanwhile the baseline executes exactly 32 control steps per call by construction while we

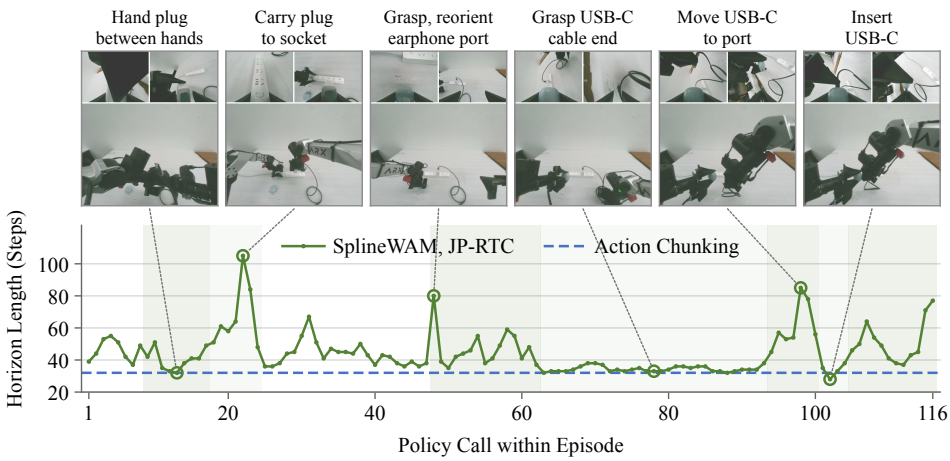


Figure 6: Horizon length at every policy call of one successful Charge Earphone trial under SplineWAM with JP-RTC. Shaded bands mark the stages we identify from the recorded observations, with a keyframe above each stage's extremum.

decode 1.2 to 1.6 times as much from a parameter window of comparable size, so the efficiency gain costs no accuracy here, as in simulation.

The two RTC variants make the sharper point, being one model under two placements of the chunk continuity constraint. Naive RTC decodes the *longer* chunks of the two on all three tasks and yet performs clearly worse on all three, in both success and progress, while what JP-RTC gives up to recover that is approximately one control step of chunk length. Constraining spline parameters directly therefore yields no gain in the length of the executed chunk, while degrading the quality of the actions within it.

Qualitative analysis. Figure 6 plots the horizon SplineWAM decodes at every policy call of one successful Charge Earphone trial. The horizon tracks the structure of the task rather than the clock, and it separates the stages into two groups. Free-space transfers, carrying the plug towards the socket and moving the USB-C connector towards the earphone port, are smooth enough for few knots to describe and decode into long chunks, several times the baseline's fixed horizon. Contact and alignment, closing the gripper on the cable end and the final insertion, attract knots densely and fall back to roughly the baseline's replanning rate, the shortest chunks of the episode arriving at the insertion itself. SplineWAM therefore replans at nearly the baseline's rate when precision demands it and stretches the horizon only where the motion is compressible, drawing its efficiency from the phases where a long chunk costs nothing, a trade a fixed horizon cannot make in either direction.

5 Conclusion

We propose SplineWAM, which replaces the fixed-length action chunk of a world action model with a fixed-size window of cubic B-spline knot times and control points. The window is fitted adaptively, so the temporal resolution of the prediction follows the complexity of the motion. Its knot times also carry the video supervision and the execution schedule, making the executed span and the replanning interval properties of the prediction rather than hyperparameters, and JP-RTC makes the representation usable asynchronously. On LIBERO-Plus and RoboCasa the method improves success rate by 8.2 and 4.4 points over an action chunking WAM while cutting policy calls per episode by 22% and 26%. As policies grow past what fits on each robot and are served centrally, the call count sets how many robots a deployment can carry. The method carries over to three bimanual real-robot tasks under asynchronous execution. There it exceeds a fixed-chunk baseline on four of six accuracy measures and matches it on a fifth, while decoding 1.2 to 1.6 times as much executed motion per call. The decoded horizon lengthens through free-space transfers and contracts at insertion, without any explicit instruction to do so.

AI USE STATEMENT

We have not used generative AI tools for any task requiring disclosure, including implementing methods, generating synthetic data, developing theoretical models or conceptual frameworks, formulating mathematical claims or writing proofs, proposing hypotheses, designing or giving feedback on experiments, cleaning datasets, and interpreting results. Additionally, we used generative AI tools to edit the paper for readability, to identify and summarise related literature, and to format tables and references. We have reviewed all AI-assisted work: AI-edited prose was checked against the underlying results, and every reference surfaced by an AI-assisted search was read in the original before being cited. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies action representations for robot manipulation. The simulated evaluation uses established public benchmarks. The real-robot evaluation runs on the authors' own laboratory hardware on three manipulation tasks with everyday objects, and the demonstrations behind it were collected in house; they involve no human subjects and contain no personally identifying information. We foresee no direct ethical concerns beyond those generally applicable to robot learning research.

REPRODUCIBILITY STATEMENT

Section 4.1 specifies the backbone, benchmarks, window sizes and fitting mode for every comparison, and Section 3.4 the closed-loop execution rule and the row budget behind every reported call count. Appendix E derives the JP-RTC update in full and Appendix D records data construction, the parameter window, its scaling and decoding safeguards, masking, model dimensions and hyperparameters; Appendix B gives the real-robot platforms, action spaces and control rate.

REFERENCES

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *Advances in Neural Information Processing Systems*, volume 38, 2025.

Qingwen Bu, Yanting Yang, Jisong Cai, Shenyuan Gao, Guanghui Ren, Maoqing Yao, Ping Luo, and Hongyang Li. Univla: Learning to act anywhere with task-centric latent actions. *arXiv preprint arXiv:2505.06111*, 2025.

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

Xiaoyu Chen, Hangxing Wei, Pushi Zhang, Chuheng Zhang, Kaixin Wang, Yanjiang Guo, Rushuai Yang, Yucen Wang, Xinquan Xiao, Li Zhao, Jianyu Chen, and Jiang Bian. villa-x: Enhancing latent action modeling in vision-language-action models. *arXiv preprint arXiv:2507.23682*, 2025.

Paul Dierckx. *Curve and surface fitting with splines*. Oxford University Press, 1995.

Zibin Dong, Yicheng Liu, Shiduo Zhang, Baijun Ye, Yifu Yuan, Fei Ni, Jingjing Gong, Xipeng Qiu, Hang Zhao, Yinchuan Li, and Jianye Hao. Actioncodec: What makes for good action tokenizers. *arXiv preprint arXiv:2602.15397*, 2026.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, et al. Libero-plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

Jun Guo, Qiwei Li, Peiyan Li, Zilong Chen, Nan Sun, Yifei Su, Heyun Wang, Yuan Zhang, Xinghang Li, and Huaping Liu. Unified 4d world action modeling from video priors with asynchronous denoising. *arXiv preprint arXiv:2604.26694*, 2026.

Xiaoshen Han, Haoyu Xiong, Haonan Chen, Chaoqi Liu, Antonio Torralba, Yuke Zhu, and Yilun Du. B-spline policy: Accelerating manipulation policies via b-spline action representations. *arXiv preprint arXiv:2607.09648*, 2026.

Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior generation with latent actions. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 26991–27008, 2024.

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

Yuanchang Liang, Xiaobo Wang, Kai Wang, Shuo Wang, Xiaojiang Peng, Haoyu Chen, David Kim Huat Chua, and Prahlad Vadakkepat. Adaptive action chunking at inference-time for vision-language-action models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

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

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

Chaoqi Liu, Xiaoshen Han, Jiawei Gao, Yue Zhao, Haonan Chen, and Yilun Du. Oat: Ordered action tokenization. *arXiv preprint arXiv:2602.04215*, 2026.

Yicheng Liu, Shiduo Zhang, Zibin Dong, Baijun Ye, Tianyuan Yuan, Xiaopeng Yu, Linqi Yin, Chenhao Lu, Junhao Shi, Luca Jiang-Tao Yu, Liangtao Zheng, Tao Jiang, Jingjing Gong, Xipeng Qiu, and Hang Zhao. Faster: Toward efficient autoregressive vision language action modeling via neural action tokenization. *arXiv preprint arXiv:2512.04952*, 2025.

Huaihai Lyu, Chaofan Chen, Senwei Xie, Pengwei Wang, Xiansheng Chen, Shanghang Zhang, and Changsheng Xu. Omnisat: Compact action token, faster auto regression. *arXiv preprint arXiv:2510.09667*, 2025.

Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In *Conference on Robot Learning*, 2023.

Atharva Mete, Haotian Xue, Albert Wilcox, Yongxin Chen, and Animesh Garg. Quest: Self-supervised skill abstractions for learning continuous control. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

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

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

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. Fast: Efficient action tokenization for vision-language-action models. In *Robotics: Science and Systems XXI*, 2025.

Jiaming Song, Arash Vahdat, Morteza Mardani, and Jan Kautz. Pseudoinverse-guided diffusion models for inverse problems. In *International Conference on Learning Representations*, 2023.

Bahey Tharwat, Yara Nasser, Ali Abouzeid, and Ian Reid. Latent action pretraining through world modeling. *arXiv preprint arXiv:2509.18428*, 2025.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17(3):261–272, 2020. doi: 10.1038/s41592-019-0686-2.

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

Hongtao Wu, Ya Jing, Chilam Cheang, Guangzeng Chen, Jiafeng Xu, Xinghang Li, Minghuan Liu, Hang Li, and Tao Kong. Unleashing large-scale video generative pre-training for visual robot manipulation. In *International Conference on Learning Representations*, volume 2024, pp. 10641–10662, 2024.

Angen Ye, Boyuan Wang, Chaojun Ni, Guan Huang, Guosheng Zhao, Hao Li, Hengtao Li, Jie Li, Jindi Lv, Jingyu Liu, Min Cao, Peng Li, Qiuping Deng, Wenjun Mei, Xiaofeng Wang, Xinze Chen, Xinyu Zhou, Yang Wang, Yifan Chang, Yifan Li, Yukun Zhou, Yun Ye, Zhichao Liu, and Zheng Zhu. Gigaworld-policy: An efficient action-centered world–action model. *arXiv preprint arXiv:2603.17240*, 2026a.

Seonghyeon Ye, Joel Jang, Byeongguk Jeon, Sejune Joo, Jianwei Yang, Baolin Peng, Ajay Mandlekar, Reuben Tan, Yu-Wei Chao, Bill Yuchen Lin, Lars Liden, Kimin Lee, Jianfeng Gao, Luke Zettlemoyer, Dieter Fox, and Minjoon Seo. Lapa: Latent action pretraining from videos. *arXiv preprint arXiv:2410.11758*, 2024.

Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, Ayaan Malik, Kyungmin Lee, William Liang, Nadun Ranawaka, Jiasheng Gu, Yinzhen Xu, Guanzhi Wang, Fengyuan Hu, Avnish Narayan, Johan Bjorck, Jing Wang, Gwanghyun Kim, Dantong Niu, Ruijie Zheng, Yuqi Xie, Jimmy Wu, Qi Wang, Ryan Julian, Danfei Xu, Yilun Du, Yevgen Chebotar, Scott Reed, Jan Kautz, Yuke Zhu, Linxi Fan, and Joel Jang. 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.

Qihang Zhang, Lin Li, Luyao Zhang, Shuai Yang, Yiming Luo, Shuaiting Li, Ruilin Wang, Junke Wang, Jiahao Shao, Gangwei Xu, et al. Native video-action pretraining for generalizable robot control. *arXiv preprint arXiv:2607.08639*, 2026.

Tian-Tsong Zhang, David Pfrommer, Chenyang Pan, Nikolai Matni, and Max Simchowitz. Action chunking and exploratory data collection yield exponential improvements in behavior cloning for continuous control. *arXiv preprint arXiv:2507.09061*, 2025.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems XIX*, 2023. doi: 10.15607/RSS.2023.XIX.016.

Hongyi Zhou, Weiran Liao, Xi Huang, Yucheng Tang, Fabian Otto, Xiaogang Jia, Xinkai Jiang, Simon Hilber, Ge Li, Qian Wang, Ömer Erdinç Yağmurlu, Nils Blank, Moritz Reuss, and Rudolf Lioutikov. Beast: Efficient tokenization of b-splines encoded action sequences for imitation learning. *arXiv preprint arXiv:2506.06072*, 2025.

Pengfei Zhou, Shengcong Chen, Di Chen, Jiaxu Wang, Rongjun Jin, Bingwen Zhu, Yike Pan, Songen Gu, Kuanning Wang, Shufeng Nan, Xingyu Qiu, Chenhao Qiu, Pu Yang, Yunuo Cai, Jianxiong Gao, Yifan Li, Yanwei Fu, Xiangyu Yue, Zhi Chen, and Jianlan Luo. τ_0-wm: A unified video-action world model for robotic manipulation. *arXiv preprint arXiv:2606.01027*, 2026.

APPENDIX

A Additional results **14**

B Real-robot tasks **15**

C Asynchronous execution in simulation **16**

D Implementation details **17**

E Derivation of the JP-RTC update **19**

F Limitations **22**

A ADDITIONAL RESULTS

This appendix collects the results deferred from Section 4: the per-category breakdowns behind the summary of Table 1, the synchronous experiments on the original LIBERO suite, and the LIBERO-Plus counterpart of the asynchronous comparison reported on RoboCasa in the main text. Protocols, baselines and efficiency measures are exactly those of Section 4.1.

Per-category breakdowns. Table 3 and Table 4 expand the two suites of Table 1 into their task categories. The category columns show that the gains of SplineWAM are not carried by one factor or family: on LIBERO-Plus, SplineWAM-Knot leads the action chunking baseline on six of the seven perturbations, with camera viewpoint and language instruction the largest margins at 11.0 and 19.4 points, and light conditions the single factor where it does not; on RoboCasa it leads on all four families. The two alignment variants are close in most columns and the summary margin of 1.2 and 1.6 points comes from a few: knot-grid alignment gains 7.5 points on background textures and 4.2 on turning tasks, while giving back 2.5 points on camera viewpoint.

Table 3: LIBERO-Plus success rate (%) by action representation, at $\varepsilon = 0.01$ under synchronous execution, expanding Table 1. Perturbation factors are background textures (BT), camera viewpoints (CV), language instructions (LI), light conditions (LC), objects layout (OL), initial states (IS) and sensor noise (SN); averages are episode-weighted.

	Task Performance								Efficiency	
Action representation	BT	CV	LI	LC	OL	IS	SN	Average	Calls/episode	Steps/call
Action Chunking	61.1	43.5	70.8	87.4	75.9	66.6	63.8	66.3	33.6	8.00
BEAST	65.1	43.1	65.3	89.8	75.8	61.8	71.6	66.6	35.5	8.00
Naive B-Spline w/o video	55.6	32.9	71.0	85.1	66.7	53.7	61.0	60.0	38.7	7.92
Naive B-Spline	58.7	51.4	82.4	89.5	77.8	73.2	64.4	70.7	27.3	8.36
SplineWAM-Window	54.7	57.0	87.8	84.2	79.1	73.7	74.3	73.3	27.1	8.39
SplineWAM-Knot	62.2	54.5	90.2	87.3	80.1	73.5	74.3	**74.5**	**26.2**	**8.41**

Table 4: RoboCasa success rate (%) by action representation at $\varepsilon = 0.08$ under synchronous execution, expanding Table 1 under the same protocol and efficiency measures as Table 3. Columns are task families; averages are episode-weighted.

	Task Performance					Efficiency	
Action representation	Pick-and-place	Open/close	Turn	Coffee	Average	Calls/episode	Steps/call
Action Chunking	53.1	89.7	72.7	62.0	69.1	21.9	16.00
BEAST	44.1	88.7	71.0	59.7	65.0	24.6	16.00
Naive B-Spline w/o video	34.2	66.7	65.7	62.3	55.0	32.7	14.26
Naive B-Spline	52.8	91.8	69.6	69.3	69.5	16.9	17.98
SplineWAM-Window	55.2	95.3	72.1	69.0	71.9	17.5	18.10
SplineWAM-Knot	56.1	94.7	76.3	71.3	**73.5**	**16.3**	**18.18**

Original LIBERO. Table 5 reports the four standard LIBERO task suites under synchronous execution; we did not run the asynchronous comparison on this suite. We place these results here rather than in the main text because success rates on unperturbed LIBERO are high and tightly clustered, with every representation that is not actively broken landing within about a point and a half of the others and SplineWAM-Knot only 0.1 points above the action chunking baseline, so the suite separates the comparisons far less than LIBERO-Plus does, and conclusions drawn from it would rest on differences smaller than single-run noise. Two observations survive that compression of the scale. Removing the auxiliary video term is the one change that clearly hurts (90.5 against 95.7 for the same representation with the term), and the efficiency ordering matches the other suites, with SplineWAM-Knot issuing 17.2 calls per episode against 19.8 for action chunking.

The tolerance sweep is the one place where this suite disagrees with the perturbed ones: $\varepsilon = 0.005$ reaches 98.0 against 96.4 at $\varepsilon = 0.01$, so on unperturbed LIBERO the least compressed represen-

tation is the stronger one, whereas on RoboCasa and LIBERO-Plus a moderate tolerance is slightly better.

Table 5: Original LIBERO success rate (%) across the synchronous experiments of Section 4, with blocks matching Table 3 and Figure 4. Efficiency was instrumented for a subset of the representations only; blank entries were not measured.

	Task Performance					Efficiency	
Setting	Spatial	Object	Goal	Long	Average	Calls/episode	Steps/call
Action representation (at $\varepsilon = 0.01$)							
Action Chunking	97.4	99.4	95.0	93.2	96.3	19.8	8.00
BEAST	95.4	98.4	94.4	90.8	94.8	20.2	8.00
Naive B-Spline w/o video	86.6	96.8	94.2	84.2	90.5	21.6	7.15
Naive B-Spline	98.2	98.4	95.2	91.0	95.7	17.6	8.76
SplineWAM-Window	98.2	98.4	96.2	91.0	96.0	17.5	8.78
SplineWAM-Knot	99.0	99.0	96.6	91.0	**96.4**	**17.2**	**8.82**
Fitting tolerance							
$\varepsilon = 0.005$	99.2	99.8	97.0	96.2	**98.0**		
$\varepsilon = 0.01$	99.0	99.0	96.6	91.0	96.4		
$\varepsilon = 0.025$	94.4	97.2	90.4	81.0	90.8		
$\varepsilon = 0.05$	79.8	73.0	79.0	66.8	74.7		

B REAL-ROBOT TASKS

This appendix describes the three real-robot tasks of Section 4.4 and the protocol used to score them.

Arrange Bookshelf. Books scattered across a desk and on the shelf must be placed upright and aligned in the shelf. Several consecutive placements, each moving between desk and shelf, make this a long-horizon mobile manipulation task.

Charge Earphone. The robot inserts a charging plug into a wall socket and connects the earphone case to the cable. Both insertions demand tight alignment at the connector, making this a high-precision manipulation task.

Clean Whiteboard. The robot erases writing from a whiteboard and returns the eraser to its holder. Judging which regions are still marked requires visual understanding, but the wiping strokes themselves are highly repetitive and therefore highly compressible.

Platform and action space. Charge Earphone and Clean Whiteboard run on an ARX AC One, a bimanual platform of two 6-DoF arms on a fixed base; Arrange Bookshelf runs on a Lift-2S, the same bimanual upper body on a 3-DoF mobile base with an additional prismatic waist. Observations are three 240×320 RGB streams from a head camera and one camera per wrist, and the control loop runs at 30 Hz. An action is $d = 20$ dimensional on the two fixed-base tasks, comprising per arm 3 end-effector translation, 6 rotation in the two-column representation and 1 gripper, and $d = 24$ on Arrange Bookshelf, which adds base velocity and the waist. The parameter window keeps the 36 rows of the simulated setting, and the constrained prefix again covers $|S| = 8$ control steps. The representation is full-dimensional continuous cubic B-splines on the real robot as in simulation, gripper dimensions included.

Protocol. Every method runs asynchronously, since a real deployment cannot pause the robot for inference. We score each trial against the annotated task nodes of Table 6. Arrange Bookshelf is scored on the three placements it consists of, one node per book shelved, so each node is worth 33.3%. The other two tasks are scored on four nodes worth 25% each, the stages on which the methods separate: every method reaches the earlier stages in nearly every trial, and the stages after the last node succeed or fail exactly with it. Progress (PG) is the fraction of a task's nodes that a trial reaches, averaged over trials, and success (SR) requires all of them. We run 10 scene configurations

per task and 3 trials per configuration, giving 30 trials per method per task; configurations vary the initial pose and the objects involved, such as the orientation of the earphone charging port and the colour of the marker to be erased.

Table 6: The scored nodes of each task, in order. Progress credits an equal share per node reached: 33.3% for Arrange Bookshelf and 25% for the other two.

Task	Node 1	Node 2	Node 3	Node 4
Arrange Bookshelf	first book shelved	second book shelved	third book shelved	—
Charge Earphone	grasp plug	plug into socket	grasp USB-C	insert USB-C
Clean Whiteboard	erase two thirds	nearly all erased	fully erased	eraser returned

C ASYNCHRONOUS EXECUTION IN SIMULATION

The real-robot comparison of Section 4.4 runs every method asynchronously. This appendix isolates the same comparison in simulation under a controlled inference delay, where the synchronous references are available alongside it. The delay is injected as a fixed 8 control steps, matching the constrained prefix $|S| = 8$: the environment advances eight steps on the previous plan while the next is being computed, so every call arrives with exactly the overlap on which the continuity residual is defined. Fixing the delay in this way renders the three asynchronous rows comparable at a single operating point rather than averaged over a latency distribution, at the cost of not resolving how the gap scales with the delay, which Appendix F records as open.

Table 7: RoboCasa success rate (%) under synchronous and asynchronous execution with a controlled inference delay. The asynchronous rows sit below the synchronous references by construction, so the comparison of interest is among the three of them, which differ in where the continuity residual is defined.

Model & inference setting	Pick-and-place	Open/close	Turn	Coffee	Average
Action Chunking, sync	53.1	89.7	72.7	62.0	69.1
SplineWAM, sync	56.1	94.7	76.3	71.3	**73.5**
Action Chunking, async RTC	42.6	88.8	70.4	65.3	65.1
SplineWAM, async naive RTC	40.5	88.3	70.3	61.3	63.8
SplineWAM, async JP-RTC	43.5	87.7	73.3	63.3	**65.7**

Asynchronous execution trades accuracy for the removal of inference latency from the control loop. Real-time chunking is a method for keeping that trade acceptable, not for improving success rate, so the comparison that matters is among the three asynchronous rows of Table 7.

Applying ordinary real-time chunking directly in spline parameter space is actively harmful. SplineWAM with naive RTC lands 1.3 points *below* the action chunking baseline under the same asynchronous protocol: the representation that was the stronger of the two synchronously, by 4.4 points, becomes the weaker one asynchronously. This is the failure mode predicted in Section 3.4 and drawn in Figure 3.

JP-RTC recovers this. It improves over naive RTC by 1.9 points, bringing SplineWAM back above the action chunking baseline. Defining the residual on decoded actions and pulling it back through the decoder Jacobian is thus what makes a spline representation usable asynchronously at all: without it the representation is a liability under latency, and with it it is at least as good as a per-timestep chunk while retaining the temporal compression and the reduced call counts of Table 1. A gap to synchronous execution remains, 7.8 points, larger than the 4.0 points that separate synchronous from asynchronous action chunking; Appendix F takes up what this leaves open.

Table 8 is the LIBERO-Plus counterpart and reproduces the same conclusion. The two suites differ in the size of the asynchronous gap, which is far larger on LIBERO-Plus: JP-RTC reaches 47.5% there against a synchronous 74.5%, a 27.0-point drop where RoboCasa loses 7.8. The ordering among the asynchronous rows is nonetheless the same, and the margin JP-RTC opens over naive

spline-space RTC is wider, 8.1 points against 1.9; it also clears the action chunking baseline under the same protocol by 2.7 points.

Table 8: LIBERO-Plus success rate (%) under synchronous and asynchronous execution, under the same protocol as Table 7. Columns follow Table 3.

Model & inference setting	BT	CV	LI	LC	OL	IS	SN	Average
Action Chunking, sync	61.1	43.5	70.8	87.4	75.9	66.6	63.8	66.3
SplineWAM, sync	62.2	54.5	90.2	87.3	80.1	73.5	74.3	**74.5**
Action Chunking, async RTC	30.9	15.6	50.5	78.7	56.4	42.4	44.8	44.8
SplineWAM, async naive RTC	20.1	17.1	53.7	51.6	53.3	35.0	43.0	39.4
SplineWAM, async JP-RTC	25.7	27.9	65.5	61.7	59.7	39.3	50.5	**47.5**

D IMPLEMENTATION DETAILS

Our codebase is derived from the public release of Fast-WAM (Yuan et al., 2026), which supplies the two-expert backbone, the Wan VAE (Wan et al., 2025) video encoder, the flow-matching training loop and the evaluation harness. We keep these unchanged and add three components: the video–action alignment of Section 3.3, the B-spline action space that replaces the raw action chunk as the action expert's prediction target, and JP-RTC in the asynchronous inference path. The baselines in Section 4.1 are the same codebase with the corresponding component disabled.

Backbone and training objective. SplineWAM is obtained by fine-tuning the Wan2.2-5B video generation model (Wan et al., 2025) under the two-expert arrangement of Fast-WAM, in which a video expert and an action expert are trained jointly. The observation frames of a sample are encoded to latents by the Wan VAE, and the video expert predicts a flow target on those latents while the action expert predicts one on the B-spline parameter window. Two flow targets are therefore predicted per sample and the losses are summed as $\mathcal{L} = \lambda_v \mathcal{L}_{\text{video}} + \lambda_a \mathcal{L}_{\text{action}}$ with $\lambda_v = \lambda_a = 1$, with per-sample scheduler weights applied to each term. The action loss is a mean squared error on the flow target in the model's normalised output space, computed over the whole parameter window, both parameter blocks being normalised to the common scale described below.

Adaptive knot fitting uses the FITPACK implementation shipped with SciPy (Dierckx, 1995; Virtanen et al., 2020): we call its least-squares spline routine with an explicit knot vector and drive the knot-insertion loop of Section 3.2 ourselves, so that the stopping rule is the elementwise maximum of Equation 2 in normalised action units rather than SciPy's own smoothing criterion. Fitting is done offline as a dataset preprocessing step; the resulting parameter windows are cached alongside the demonstrations, so training sees no fitting cost.

Fixed-size local windows. Following Han et al. (2026), the fixed-size window of Section 3.2 is a contiguous run of knots and control points of the full-episode spline, extended with boundary support so that the curve remains evaluable across the interval the window is meant to cover, and we use $L_{\text{BSP}} = L_{\text{video}} + 3$ rows for an observation window of L_{video} frames. The $+3$ is a budget we chose rather than an identity derived from k, so we set out how the 36 rows of our configuration are allocated. A row carries a knot time and a control point together, and the two are consumed differently. On the knot side the boundary is symmetric: with knots $u_0, \ldots, u_{35}$ the cubic basis is fully supported only on $[u_3,\ u_{32}]$, so three knots at each end, six in all, serve as boundary support and the remaining thirty knots, spanning twenty-nine intervals, carry the evaluable curve. On the control-point side it is not. A basis function $N_{j,3}$ is supported on $[u_j,\ u_{j+4}]$, so c_0, c_1, c_2 are active on the first evaluable spans and are ordinary parameters, whereas $c_{32}, \ldots, c_{35}$ are supported only at or beyond u_{32} and contribute nothing to the decoded trajectory: the window has 32 effective control points, matching the 32-step chunk of the baselines.

The trailing four are therefore structural padding rather than free capacity, and we keep them so that every row has the same layout. They are still supervised, since the parameter-space flow-matching loss described above runs over the whole window, and their knot times are not inert: u_{32} is the right endpoint of the evaluable interval and so sets $t_{\max}$ and the decoded length M, while u_{33}, u_{34}, u_{35} are what the multiplicity trimming below acts on. Only the four tail control points are without effect.

Each row stores one absolute knot time alongside one control point, so a row is $1 + d$ dimensional, $[\,t,\ c\,]$ with $c \in \mathbb{R}^d$, for actions of any dimension d the embodiment provides, where t is the knot time *relative to the current observation*, measured in data frames: an absolute position on the local time axis, not an interval between neighbouring knots. Windows are anchored by assigning each observation to the spline span containing it and subtracting the observation frame index from the global knot times. Since boundary support consumes rows at both ends of the evaluable interval, the number of rows exceeds the number of effective knots, and we speak of a fixed-size *parameter window* rather than a fixed number of tokens. L_{BSP} and L_{video} count different quantities, parameter rows in the one case and observation frames in the other, and the relation between them is a budget chosen so that a window spans a comparable stretch of the episode to the observation it is anchored to. Adaptive fitting decouples the two: at a fixed L_{BSP} the duration those rows describe varies with the motion, which is the property the representation is built to provide.

Parameter scaling. The two blocks of a row are not naturally on the same scale: control points inherit the $[-1, 1]$ normalisation of the action space, whereas a knot time is a frame offset that can run to a hundred frames or more. Both are therefore mapped to a common range before they reach the model, knot times by an affine map fitted over the dataset to the interval the local windows occupy, so that the action loss weights a knot-time residual and a control-point residual comparably and the unbounded coordinate cannot dominate the gradient. The model predicts, and JP-RTC operates in, these normalised coordinates throughout; decoding inverts the map before evaluating Equation 1, and the knot times quoted in the paper are in frames.

Decoding safeguards. A predicted window may not be a well-formed spline, so before evaluating one the decoder makes the knot times non-decreasing with a running maximum. Indexing the rows $j = 0, \ldots, L_{\mathrm{BSP}} - 1$, it then restricts evaluation to $[\,u_k,\ u_{L_{\mathrm{BSP}}-1-k}\,]$, the interval over which the cubic basis is fully supported, so the k knot times at each end serve as boundary support only. It also trims surplus repeated endpoint knots whose multiplicity would exceed what a cubic spline admits.

The running maximum requires a caveat, since it repairs at decode time what the parameterisation does not guarantee. Flow matching generates in an unconstrained $\mathbb{R}^{L_{\mathrm{BSP}} \times (1+d)}$ and cannot enforce $t_j \le t_{j+1}$, and the repair proceeds by setting an out-of-order knot equal to its predecessor rather than by reordering, so an inversion becomes a coincident interior knot, which reduces the curve from C^2 to C^1 at that point and further still if several knots coincide. In practice the fitted windows the model is trained on are monotone by construction and the predictions inherit this closely enough that the safeguard is seldom active, and the seam smoothing applied on the real robot removes what remains; the guarantee is nonetheless a decode-time clamp rather than a property of the representation. Predicting positive increments $\Delta u_j > 0$ and accumulating them would make monotonicity structural, at the cost of turning knot times into a quantity whose errors compound along the window, and whose absolute anchoring to the current observation would have to be recovered by summation. That anchoring is relied upon both by the alignment of Section 3.3 and by the target shift of Section 3.4. We compared the two parameterisations in preliminary experiments and found absolute times the better of the two, and we retain them for that reason.

Padding and masking. Unlike a raw action chunk near an episode boundary, the action branch carries no padding mask: a tail window is completed by repeating the final control point, which is a well-formed spline rather than a placeholder, and every row is supervised. The video branch does carry one, because knot-grid alignment selects frames at knot offsets and a window near the end of an episode may request frames beyond it; repeated tail knots then select the same frame more than once and are marked as padding. Those positions are excluded from $\mathcal{L}_{\mathrm{video}}$, so that no gradient is taken against a duplicated target, and they are masked out of attention as keys, so that neither branch attends to them, while remaining present as queries in order to keep the latent grid at a fixed shape. Action parameters attend to the unpadded video positions under the backbone's mask, unchanged from Fast-WAM.

The gripper dimension. The gripper command enters as one more continuous coordinate of the same spline, so a demonstration's open–close transition is a steep ramp rather than a step, and the decoded value is thresholded at the midpoint of the normalised range to recover a binary command, with no hysteresis. This has two consequences. Overshoot around a transition is harmless here, since the threshold discards it provided the sign is correct, and the low-level controller saturates the

command in any case. The knot budget is affected more substantially: a transition in the gripper channel is precisely the kind of local feature to which the ℓ_∞ criterion of Equation 2 responds, and because all dimensions share one knot vector, knots inserted to resolve it are expended on the entire row. This is a genuine cost of the shared knot vector rather than a defect of the gripper's encoding, and it contributes to the decline in achievable compression as the action space widens, as Appendix F discusses. It is mitigated in practice by the fact that gripper transitions coincide with the contact phases that attract knots in any event.

Window sizes and action dimensions. Observation windows use $L_{\text{video}} = 33$ frames with an action–video frequency ratio of 4, giving 9 supervised video frames and a 36-row B-spline window against the 32-step raw action chunk of the baselines. Both suites drive a single arm with $d = 7$ dimensional actions (3 translation, 3 axis-angle, 1 gripper), so a parameter row is 8 dimensional and a window has $n = 288$ parameters; the constrained prefix of JP-RTC covers $|S| = 8$ control steps, giving $m_S = 56$. The representation is full-dimensional continuous cubic B-splines throughout; we do not use zero-order-hold gripper variants. At the row budgets of Section 4.1 the baselines execute 8 and 16 steps of their 32-step chunk under the same protocol.

Training and evaluation protocol. The 24 RoboCasa tasks of Section 4.1 are grouped in our tables into four families: pick-and-place, opening and closing articulated objects, turning knobs and levers, and operating a coffee machine. Training uses programmatically generated trajectories and evaluation is on unseen layouts. On LIBERO we train for 10 epochs at a global batch size of 128. On RoboCasa we train for 4 epochs at a global batch size of 256, using 300 demonstrations per task generated with MimicGen (Mandlekar et al., 2023). Both use a peak learning rate of 10^{-4} with a decaying schedule, and every method in a given comparison is trained under the identical budget, sharing the demonstrations, the action normalisation and the low-level controller, so that a comparison changes only the action representation or the video–action alignment. Reported numbers are single-run point estimates.

The LIBERO-Plus results are zero-shot: we train no model on LIBERO-Plus. Each LIBERO-Plus number is produced by taking the model trained on the LIBERO training split and evaluating it under the seven perturbation factors without any adaptation, fine-tuning, or perturbation-specific model selection. The comparison in Section 4.2 is therefore a generalisation test rather than an in-distribution one, which is what makes the margins there larger than on the unperturbed suite: all representations fit the LIBERO training distribution well enough that success rates saturate, and they separate only once the observation is displaced.

E Derivation of the JP-RTC update

This appendix derives Equation 3 from the RTC update it generalises. With A_t^τ the action chunk at flow-matching time τ, Y the guidance target formed from the previous chunk, and W the mask that weights the frozen prefix at 1 and decays over the overlap region, Black et al. (2025) correct the sampler's velocity by

$$v_{\Pi\text{GDM}}(A_t^\tau, o_t, \tau) = v(A_t^\tau, o_t, \tau) + \min\left(\beta, \frac{1-\tau}{\tau \cdot r_\tau^2}\right) \left(Y - \widehat{A}_t^1\right)^\top \operatorname{diag}(W) \frac{\partial \widehat{A}_t^1}{\partial A_t^\tau}, \tag{4}$$

$$\text{where} \quad \widehat{A}_t^1 = A_t^\tau + (1-\tau) v(A_t^\tau, o_t, \tau), \qquad r_\tau^2 = \frac{(1-\tau)^2}{\tau^2 + (1-\tau)^2}, \tag{5}$$

and $\widehat{A}_t^1$ is the one-step estimate of the clean chunk, the weight being clipped at β for stability at the small step counts used in control. Throughout, τ is the flow-matching time, z_t^τ the parameter window, $\widehat{z}_t^1$ its one-step estimate, $n = \dim z$, and $m_S = |S| \cdot d$ the number of constrained raw-action scalars ($m_S = 56$ against $n = 288$ in our configuration, though only a local subset of those n parameters acts on the prefix, as the damping paragraph below makes precise). Integer control-step offsets are written s, reserving τ for the flow-matching time throughout. We derive the update under RTC's hard mask, so that S is exactly the constrained set and $W_S = I$; the soft mask of Black et al. (2025) enters at the end as a weighting of the residual. The guidance target is the previous parameter window Y_z, and the robot has committed to the segment of it that overlaps the arriving window, $y_S = D_{S+\Delta t}(Y_z)$ for an inference latency of Δt control steps, the shift placing the two

windows on a common time origin at the current observation; equivalently y_S may be read off the controller's buffer of committed actions directly, which is what the implementation does. We write $e_S = y_S - D_S(\widehat{z}_t^1) \in \mathbb{R}^{m_S}$ for the unweighted residual and $r_S = W_S e_S$ for its masked counterpart, with $W_S = \mathrm{diag}(w_S) \otimes I_d$ lifting the per-step weights $w_S \in \mathbb{R}^{|S|}$ to the m_S action scalars. Because the residual is formed after decoding, the mask indexes control steps rather than parameter rows and acts identically on all d dimensions of a step, which is the first of the two differences from the RTC update; the second is the pullback itself. Everything below is in the model's normalised coordinates, as in our implementation: z is the parameter window the sampler generates, D maps it to actions in normalised units, and J_S is the Jacobian in those coordinates.

The guidance term is already a damped pseudoinverse. Pseudoinverse guidance (Song et al., 2023) treats chunk continuity as an inverse problem $Y = \mathcal{H}(A_t^1) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_y^2 I)$, and approximates the posterior over clean samples by an isotropic Gaussian centred on the one-step estimate,

$$p_\tau(A_t^1 \mid A_t^\tau) \approx \mathcal{N}\big(\widehat{A}_t^1,\ r_\tau^2 I\big), \tag{6}$$

with r_τ^2 as in Equation 5. For a linear $\mathcal{H}(A) = HA$ the pushforward of Equation 6 is again Gaussian, $\mathcal{N}(H\widehat{A}_t^1,\ r_\tau^2 HH^\top + \sigma_y^2 I)$, giving the closed-form guidance score

$$\nabla_{A_t^\tau} \log p_\tau(Y \mid A_t^\tau) \approx \big(Y - H\widehat{A}_t^1\big)^\top \big(r_\tau^2 HH^\top + \sigma_y^2 I\big)^{-1} H \frac{\partial \widehat{A}_t^1}{\partial A_t^\tau}. \tag{7}$$

The matrix inverted in Equation 7 is a *damped* Gram matrix: factoring out r_τ^{-2}, which Equation 4 already carries in its guidance weight, leaves the Tikhonov pseudoinverse $H^\top(HH^\top + \lambda I)^{-1}$ with

$$\lambda = \sigma_y^2 / r_\tau^2. \tag{8}$$

The damped solve is therefore the exact form of the guidance term, with $\lambda \to 0$ encoding the belief $\sigma_y = 0$ that the target is matched exactly. This reading fixes the meaning of λ but not its value, which we set from the spectrum of J_S as Equation 12 describes.

Replacing the mask with a decoder. For SplineWAM the generated variable is z but the quantity that must match the previous chunk is the decoded prefix, so $\mathcal{H}(z) = D_S(z)$, which is nonlinear. Song et al. (2023) handle nonlinear measurements by supplying a map $\mathcal{H}^\dagger$ with $\mathcal{H}(\mathcal{H}^\dagger(\mathcal{H}(z))) = \mathcal{H}(z)$, as they do for quantisation and JPEG, but D_S admits no such closed form: inverting it means choosing both control points and the knot times that reparameterise them, and the least-squares fit that would do so is itself iterative. We therefore linearise at the posterior centre,

$$D_S(\widehat{z}_t^1 + \delta) = D_S(\widehat{z}_t^1) + J_S\delta + O(\|\delta\|^2), \tag{9}$$

under which the pushforward of Equation 6 is again Gaussian, $\mathcal{N}(D_S(\widehat{z}_t^1),\ r_\tau^2 J_S J_S^\top + \sigma_y^2 I)$, so substituting $H \mapsto J_S$ into Equation 7 yields the bracketed update of Equation 3,

$$\delta^\star = J_S^\top \big(J_S J_S^\top + \lambda I\big)^{-1} e_S. \tag{10}$$

Equivalently $\delta^\star$ minimises $\|J_S\delta - e_S\|_2^2 + \lambda\|\delta\|_2^2$, the regularised least-squares correction that moves the decoded prefix towards the committed actions to first order. We use the dual form because it inverts an $m_S \times m_S$ matrix rather than an $n \times n$ one, the two being identical for $\lambda > 0$.

Equation 9 is to be read branch-wise: the decoding safeguards of Appendix D are non-smooth where knots coincide or a knot is clamped, and the piecewise polynomial changes form as the active span switches, so the expansion holds where these discrete states are locally constant, the generic case since the non-differentiable set has measure zero in z; elsewhere the value automatic differentiation returns is implementation-dependent. The expansion is used only locally: $\delta^\star$ corrects the velocity rather than the sample, integrated over the denoising trajectory, so the decoded prefix is drawn towards y_S across steps with the linearisation refreshed at each. The neglected term grows as $O(\|\delta\|_2^2)$ while the Tikhonov penalty acts on $\|\delta\|_2^2$, so the λ that damps ill-conditioned directions also suppresses large steps, which bounds the update without certifying the first-order model at $\delta^\star$.

Locality and cost. The Jacobian is local: cubic B-spline decoding is piecewise polynomial in the knot intervals, so Equation 3 is one damped Gauss–Newton step per denoising step, not a projection onto an action-continuity constraint set. The extra cost over ordinary RTC is therefore one Jacobian and one $m_S \times m_S$ solve per denoising step, with $m_S = |S| \cdot d$ small.

This accounting rests on one approximation, left implicit in Equation 3 and made explicit here. Written in full, the guidance term of Equation 4 carries a factor $\partial \widehat{z}_t^1 / \partial z_t^\tau$ relating the clean estimate to the current sample, and we take it to be the identity. Carried exactly, Equation 5 gives $\partial \widehat{z}_t^1 / \partial z_t^\tau = I + (1-\tau)\, \partial v / \partial z_t^\tau$, and the correction would require a vector–Jacobian product through the whole backbone at every denoising step, a cost comparable to one backward pass per step. This is not affordable in a control loop, and ordinary RTC does not incur it either. Dropping the term treats the one-step estimate as fixed for the purpose of the guidance, and is the same stop-gradient that Song et al. (2023) rely on in practice. The claim of Section 3.4 that adaptive horizons cost no extra network evaluation is therefore exact, since the horizon is read off the predicted knot times. JP-RTC does add the Jacobian and the $m_S \times m_S$ solve described above, linear algebra that is negligible against a forward pass of a video-pretrained backbone but not, strictly speaking, free.

Reduction to ordinary RTC. For a per-timestep chunk $D_S(z) = z_S$, so $J_S = P_S$ is a selection matrix with $P_S P_S^\top = I_{m_S}$, and Equation 10 becomes $(1+\lambda)^{-1} P_S^\top e_S$: the RTC update of Equation 4 scaled by $(1+\lambda)^{-1}$, which is exactly 1 as $\lambda \to 0$, and otherwise constant, since Equation 12 gives $\mathrm{tr}(J_S J_S^\top) = m_S$ and hence $\lambda = \lambda_{\mathrm{rel}}$; the factor then reparameterises β rather than changing direction. This identifies what ordinary RTC assumes when applied to spline parameters, namely $J_S = I$: that perturbing a parameter row perturbs the correspondingly indexed action and nothing else, which is false for every cubic B-spline.

Setting the damping. Local support can make J_S ill-conditioned once knots are widely spaced. Split it into knot-time and control-point blocks, $J_S = [\, J_U \; J_C \,]$. A cubic basis function is supported on four knot spans, so constrained steps falling in p consecutive spans that each contain at least one of them touch only $p+k$ control-point rows, giving $\mathrm{rank}(J_C) \le d\,(p+k) < m_S = d\,|S|$ whenever $p + k < |S|$, and then $J_C J_C^\top$ is singular. This makes precise the ambiguity to which Section 3.4 appeals, and identifies the freedom available to the pullback. The relevant dimension is not the full n but the active dimension n_{active} of the rows whose support reaches S, whose control-point part the bound above caps at $d(p+k)$. In the smooth regime where an entire $|S| = 8$ step prefix falls inside one wide span, $p = 1$ admits at most $d(1+3) = 28$ control-point directions against $m_S = 56$ constrained scalars, the remaining $n - n_{\mathrm{active}}$ parameters exerting no influence on the prefix. The pullback is then solving an underdetermined system in which λ selects the solution, and we therefore regard the damping as structural rather than as a device for absorbing residual numerical error. Writing $J_S J_S^\top = J_C J_C^\top + J_U J_U^\top$ with both terms positive semidefinite, Weyl's inequality gives

$$\lambda_{\min}\big(J_S J_S^\top\big) \;\le\; \lambda_{\min}\big(J_C J_C^\top\big) + \|J_U\|_2^2 \;=\; \|J_U\|_2^2. \tag{11}$$

The knot-time block is itself weak, since $\partial a(s_i)/\partial u_j$ vanishes unless u_j lies in the support of a basis function active at the constrained control step s_i, so for constrained steps interior to a wide span most of its entries are zero and the rest are small. A small $\|J_U\|_2$ then leaves the full Gram matrix with a small minimum eigenvalue, and both conditions hold in the smooth, widely spaced regime where continuity matters most. The spectrum is also not scale-free, since a unit perturbation of a knot time and of a control point move the decoded actions by different amounts, so an absolute λ would mean different things at different knot densities. We set it relative to the spectrum,

$$\lambda = \lambda_{\mathrm{rel}} \cdot \mathrm{tr}\big(J_S J_S^\top\big)/m_S, \tag{12}$$

and use $\lambda_{\mathrm{rel}} = 10^{-1}$ for all reported results.

Two properties of Equation 12 should be stated explicitly, since neither follows from the Gaussian reading of Equation 8. First, it is held constant across τ, whereas $\lambda(\tau) = \sigma_y^2 / r_\tau^2$ with a stationary σ_y would diverge as $\tau \to 1$, since $r_\tau^2 \to 0$ there, and would thereby suppress the guidance of its own accord near the clean end. Holding λ fixed instead amounts to an effective $\sigma_y^2(\tau) = \lambda\, r_\tau^2 = O((1-\tau)^2)$, a measurement precision that tightens as the sample is denoised, and it leaves the divergence of the prefactor $(1-\tau)/(\tau r_\tau^2)$ in Equation 3 to be controlled by the clip β rather than by the damping. We therefore regard Equation 12 as an empirical Gauss–Newton stabilisation whose

scale is fixed by the spectrum of J_S, rather than as an exact implementation of the posterior score under stationary measurement noise. Equation 8 establishes the dimensional meaning of λ, and a τ-dependent schedule interpolating between the two is a natural variant that we did not find necessary.

Second, the penalty $\lambda\|\delta\|_2^2 = \lambda(\|\delta_U\|_2^2 + \|\delta_C\|_2^2)$ is isotropic over blocks that are not commensurable: δ_U displaces knot times, measured in frames, and δ_C displaces control points, measured in normalised action units, so a unit perturbation of each is penalised equally despite acting on the curve in different ways. Combined with $\|J_U\|_2 \ll \|J_C\|_2$ from the argument above, the minimum-norm solution places its correction almost entirely in δ_C: JP-RTC reattaches a chunk by deforming the shape of the curve while leaving its timing essentially as predicted. This is the desired behaviour at a seam, since the knot times are what carry the horizon and a correction that rewrote them would trade continuity against the adaptivity the representation exists to provide. It is nonetheless a consequence of the scaling rather than a design choice, and a block-diagonal $\mathrm{diag}(\lambda_U I, \lambda_C I)$ would make the trade explicit and adjustable; we report the isotropic form.

Soft masking. The derivation above uses the hard mask. RTC's soft mask replaces the binary weights by values decaying to zero over the overlap region, which we keep, applying it to the residual,

$$\delta^\star = J_S^\top \left(J_S J_S^\top + \lambda I\right)^{-1} r_S, \qquad r_S = W_S\, e_S. \tag{13}$$

This step is a heuristic. Weights that differ across steps amount to a per-step measurement noise $\sigma_{y,i}^2 \propto W_{S,i}^{-1}$, for which Equation 7, with the same r_τ^{-2} factored out as in Equation 8, gives

$$\delta_\Sigma = J_S^\top \left(J_S J_S^\top + \Sigma_y / r_\tau^2\right)^{-1} e_S, \qquad \Sigma_y = \mathrm{diag}(\sigma_y^2), \tag{14}$$

on the same scale as Equation 10 and generalising its λ to a diagonal matrix. Since $J_S J_S^\top$ couples constrained steps off-diagonally for a spline, Equation 14 differs from Equation 13 in direction and not only in weighting. Both cost one Cholesky factorisation; we prefer Equation 13 because it reduces to RTC's own soft-masked update when $J_S = P_S$ and requires no Σ_y to be calibrated, the soft weights being an empirical device rather than a noise model we could set. We apply W_S in raw-action space, before the pullback, so a weight sets how insistently a control step should match.

The cost of this choice should be stated precisely. Equation 13 is the solution of $\min_\delta \|J_S\delta - W_S e_S\|_2^2 + \lambda\|\delta\|_2^2$, which downweights the *target* on a decaying step rather than the discrepancy at that step; the latter corresponds to the weighted least-squares form $\min_\delta \|W_S^{1/2}(J_S\delta - e_S)\|_2^2 + \lambda\|\delta\|_2^2$. For ordinary RTC the two agree in effect, since $J_S = P_S$ is orthogonal, the constrained steps decouple, and a weight tending to zero removes step i from the problem entirely. For a spline they do not agree, because overlapping basis support renders $J_S J_S^\top$ off-diagonal: as $(W_S e_S)_i \to 0$ the target at step i vanishes while $(J_S\delta)_i$ remains penalised at unit weight, so the solve is required to hold the decoded action at the far end of the mask near its unperturbed value rather than leaving it free. The decaying window is consequently stiffer at its tail than the weights suggest, and it is this stiffness that Equation 14 would relax. We retain Equation 13 for the reasons given above, and because the region it stiffens is the one the next call replaces, but a spline soft mask does not decay as cleanly as a per-timestep one.

F Limitations

Precision at contact. A cubic spline is smooth by construction, and the fit trades exactness for compression wherever the tolerance allows it. On the one real-robot task built around precision, SplineWAM reaches the baseline's success rate but not its progress (Section 4.4), and the tolerance sweep shows accuracy collapsing as soon as the knots that resolve contact are removed (Section 4.3). The representation is therefore best suited to tasks with a mix of free-space and contact-rich motion; for a task that is contact-rich throughout, the tolerance must be set tight enough that little compression remains.

Latency tolerance. JP-RTC recovers the deficit that naive spline-space chunking opens, but a gap to synchronous execution remains on both simulated suites, and it is much larger on LIBERO-Plus than on RoboCasa (Appendix C). We do not claim that spline actions are as latency-tolerant as per-timestep chunks in general, only that JP-RTC is what makes them usable asynchronously at all.

How the gap scales with the delay, and whether it can be closed by training with the asynchronous objective rather than correcting at inference time, we leave open.

One tolerance per dataset. The fitting tolerance is a single scalar chosen per benchmark and held fixed for all of its tasks and all phases within a task. Adaptivity therefore operates within a window, through knot placement, but not across tasks: a suite containing both a purely repetitive task and a purely high-precision one would be served badly by any single value. Choosing ε per task, or predicting it, is a natural extension we do not attempt.

Embodiment and action space. A single knot vector describes all action dimensions at once, so the compression a given tolerance achieves is set by the hardest dimension to approximate, and the achievable compression falls as the action space grows. Our simulated suites drive a single arm with $d = 7$ dimensional actions, whereas the real-robot tasks are bimanual, with Arrange Bookshelf adding a mobile base. This is visible in Table 2: Arrange Bookshelf decodes the shortest chunks of the three tasks despite involving the least fine manipulation, because its wider action space leaves less redundancy for one knot vector to absorb. A per-dimension or grouped knot vector would decouple this, at the cost of the single shared temporal structure the representation is built on, and we do not attempt it here. We also represent every dimension, including the binary gripper command, as a continuous cubic spline; we did not find this to require special handling, but embodiments with more discrete dimensions may.