Spiking World-Action Models for Robotic Manipulation

Mingquan Liu[1,2] Peng Zhou[2] Shanming Yao[3] Shuai Wang[1] Simo Wu[2] Malu Zhang[1]
[1]University of Electronic Science and Technology of China [2]LuxiTech
[3]Northwestern Polytechnical University

Abstract

World action models (WAMs) predict how the world responds to actions and offer a promising path toward general embodied intelligence, but they are large and run the network many times per plan, so they are hard to deploy on robots. Spiking neural networks (SNNs) compute with sparse binary spikes and use additions instead of multiplications, which saves power. However, existing WAMs use dense videos as the world state, so a spiking version would spend most of its computation on the background. We therefore introduce SpikingWAM, the first spiking WAM, which uses a sparse task skeleton as the world state. Inspired by biological motor control, which attends only to the key positions of a task, the Task Skeleton State (TSS) keeps the keypoints of the arm and the task objects, and a small spiking front end estimates the object keypoints from images. Body Frame Spike Coding (BFSC) and Grid Coincidence Attention (GCA) keep millimetre precision, and the Persistent Single Step Readout (PSR) runs the network once per plan. On the simulated benchmarks LIBERO, LIBERO-Plus and RoboTwin 2.0, SpikingWAM achieves competitive success rates with far less energy than ANN methods and surpasses all compared methods on LIBERO-Plus. Real robot experiments further confirm its effectiveness. These results show that a task skeleton lets spiking networks implement WAMs at very low energy, offering a new solution for low power embodied intelligence.

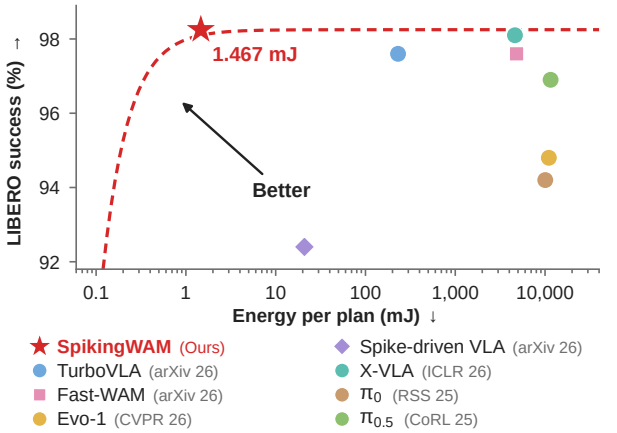


Figure 1. **Energy and success of SpikingWAM and other VLAs and WAMs on LIBERO. The upper left is better.** Energy is the estimate per plan for a 45 nm process (Sec. 5.1). The dashed line is a dividing line.

1. Introduction

World action models (WAMs) [15, 18, 29, 48, 49] let a robot learn in one model how to act and how the world changes in response. They offer a promising path toward general embodied intelligence. However, most existing WAMs use videos or visual latents as the world state [15, 18, 48, 49]. These models are large and hard to run on robots with limited resources. To this end, recent work skips future generation at inference [29, 49], which reduces computation. These models still rely on dense multiply and accumulate operations, and each plan runs the network several times for multistep sampling. A robot in closed loop control plans again and again, so this cost adds up. Lowering the energy of each plan is therefore key to deploying WAMs on board.

Neurons in spiking neural networks (SNNs) communicate with discrete spikes and fire only when the membrane potential crosses a threshold. On neuromorphic hardware, this event-driven computation replaces multiply and accumulate with accumulate, so its energy grows with the number of events [28]. SNNs have recently moved from visual perception [46] to embodied control. Spiking vision-language-action models (VLAs) [31, 38] and spiking diffusion policies [10, 35] generate actions, and a spiking world model [33] predicts the next state after an action. However, WAMs are hard to implement directly with SNNs. Mainstream WAMs are built on large video generation models [15, 18, 48, 49], and there is still no mature way to turn such models into SNNs. Even if this were done, dense video representations would still spend most events on a background unrelated to the task. Small jitter in the image would also keep triggering new events. Running the network sev-

eral times per plan further cancels the savings from sparsity. Existing approaches therefore cannot exploit the sparse and event-driven nature of SNNs. Integer spikes also struggle to keep the millimetre precision of fine manipulation.

In contrast, biological motor control attends to the key positions of a task and represents space relative to the body. When people manipulate objects, their gaze falls almost only on grasp points and targets [12]. The premotor and posterior parietal cortices represent targets relative to the arm or the hand [3, 7]. Grid cells encode position with periodic firing [8], and cortical neurons can detect spikes that arrive at the same time [16]. This suggests that the world state only needs the key positions of the arm and the task objects. Once coordinates are written into spikes relative to the body, the positions of hand and object can be compared by spike coincidence. Following the event-driven principle, only changes pass between consecutive plans.

Based on these ideas, we propose a sparse and event-driven paradigm for spiking WAMs and build SpikingWAM on it. To our knowledge, SpikingWAM is the first spiking world action model. A single spiking network trained from scratch learns future skeletons and action chunks together, and it generates only the action chunks at inference (Fig. 2). First, we propose the Task Skeleton State (TSS), which uses only the keypoints of the arm and the task objects as the world state. A small spiking perception front end estimates the object keypoints from images. Next, we propose Body Frame Spike Coding (BFSC) and Grid Coincidence Attention (GCA). BFSC encodes coordinates relative to the body and GCA compares positions by spike coincidence, so the geometry keeps millimetre precision. Finally, we propose the Persistent Single Step Readout (PSR). It runs the network once per plan and passes only the changes between consecutive plans. On LIBERO [21] and LIBERO-Plus [5], SpikingWAM reaches success rates comparable to state-of-the-art (SOTA) artificial neural network (ANN) methods. On bimanual manipulation in RoboTwin 2.0 [4], it reaches a mean success rate of 80.20% without pretraining on robot data. On LIBERO, it needs only 1.467 mJ per plan including the perception front end, which is more than an order of magnitude below every compared method (Fig. 1). On a real robot arm, SpikingWAM completes a pick and place task given by a language instruction, with a success rate comparable to ANN VLAs and only 3.07 mJ per plan. Our main contributions are as follows.

- We design TSS and a small spiking perception front end that estimates its object keypoints from images. They focus the events and attention of the policy on the parts that matter for the task, which lowers the energy of each plan.
- We propose BFSC and GCA to recover the millimetre precision that integer spikes lose. We propose PSR and an offline check that tells when it applies, so the network runs only once per plan.
- We propose a sparse and event-driven paradigm for spiking WAMs and build on it SpikingWAM, the first spiking world action model. It reaches 98.25%, 88.76% and 80.20% on LIBERO, LIBERO-Plus and RoboTwin 2.0. On LIBERO-Plus, it exceeds all compared methods.

2. Related Work

2.1. World Action Models and Fast Sampling

VLA models generate actions directly from camera images and language instructions [2, 11, 14], and compact designs further reduce the inference cost [42]. World action models build on this by adding future prediction, so that the model also learns how the scene evolves. Mainstream approaches build on pretrained video models and generate future frames together with actions [15, 18, 48], which makes them large. Recent work steadily reduces the cost of the world state. VLA-JEPA [32] replaces pixel reconstruction with visual latents. Fast-WAM [49] shows that the benefit of future prediction comes mainly from joint training, so future generation can be skipped at inference. SkeleWAM [29] represents the world state as a 3D skeleton and compresses the scene into a few keypoints. In parallel, policy distillation [26, 40] and mean flows [6, 30] compress action generation into one or a few steps. However, while these works shrink the model and the sampling steps, the computation itself still consists of dense multiply and accumulate operations. Repeated planning in closed loop control thus still costs too much energy on board.

2.2. SNNs and Spiking Embodied Models

Spiking Transformers allow SNNs to handle vision tasks at a large scale. Spiking self-attention [52], spike-driven attention [44, 45] and integer training [46] have steadily improved their scale and accuracy. This progress has carried SNNs into embodied control along two lines. One line generates actions. Spiking VLAs [31, 38] and spiking diffusion policies [10, 35] have been applied to navigation and manipulation. The other line predicts states. A spiking world model [33] learns the next state after a given action for model-based reinforcement learning. A separate network produces the actions. However, the two lines have developed separately. Action generation and scene prediction have not been unified in one spiking network, so world action models do not yet use event-driven computation.

3. Preliminaries

3.1. Spiking Neuron Model

A spiking neuron integrates its input into a membrane potential and fires a spike only when the potential crosses a threshold. The synaptic input of layer l is $\mathbf{I}^l = W^l \mathbf{s}^{l-1}$, where W^l is the synaptic weight matrix and $\mathbf{s}^{l-1}$ holds the

spikes fired by the previous layer. A neuron that does not fire triggers no synaptic computation, so the next layer only accumulates the weights of the neurons that fire. We use the bipolar integer spiking neuron [46], written $\mathcal{SN}(\cdot)$,

$$\mathbf{s}^l = \mathcal{SN}(\mathbf{u}^l) = \mathrm{clamp}\big(\mathrm{round}(\mathbf{u}^l / V_{\mathrm{th}}^l),\, -D,\, D\big), \quad (1)$$

$$\mathbf{s}^l = \textstyle\sum_{d=1}^{D} \mathbf{s}^l[d], \qquad \mathbf{s}^l[d] \in \{-1, 0, 1\}, \quad (2)$$

where $\mathbf{u}^l$ is the membrane potential, V_{th}^l is a learnable threshold, and D is the largest number of spikes that a neuron fires in one forward pass. During training the neuron outputs an integer. At inference the integer output unfolds over D time steps into unit spikes $\mathbf{s}^l[d]$ of the same sign. The next layer then only adds or subtracts one weight column for each spike. Within $[-D, D]$ the rounding passes the gradient with a straight through estimator, so the network can be trained end to end.

3.2. Spiking Flow Matching

Flow matching [20, 22] learns a velocity field that transports Gaussian noise to the data distribution. Given data x and noise $\epsilon \sim \mathcal{N}(0, I)$, the straight path $x^\tau = (1-\tau)x + \tau\epsilon$ is pure noise at $\tau = 1$. The velocity field v_θ regresses the velocity along the path,

$$\mathcal{L}_{\mathrm{FM}}(x) = \mathbb{E}_{\tau,\epsilon}\big\| v_\theta(x^\tau, \tau \mid c) - (\epsilon - x) \big\|^2, \quad (3)$$

where c is the condition. Sampling starts from $x^1 = \epsilon$ and integrates to $\tau = 0$ with N Euler steps, $x^{\tau - 1/N} = x^\tau - v_\theta(x^\tau, \tau \mid c)/N$. Spiking flow matching uses a spiking Transformer built from the neurons of Eq. (1) as v_θ. An input layer turns the noisy sample x^τ into membrane potentials. After that, the weights of the self-attention and feedforward layers read only spikes. Each integration step runs the spiking network once.

In a world action model, the condition c is the language instruction ℓ and the current world state o_t. The model applies flow matching to an action chunk A_t of H_a steps and to the future world state O_t at the same time. A readout with several samples [29] integrates k action chunks and takes the medoid chunk, which is the sample with the smallest total distance to the others. Each plan therefore runs the network kN times.

4. Method

SpikingWAM addresses the three obstacles to implementing WAMs with SNNs (Appendix A) in three stages along the data flow (Figs. 2 and 3). Section 4.1 decides the world state and writes it into spikes, Sec. 4.2 models the relative geometry of hand and object, and Sec. 4.3 reads out the action chunk. Section 4.4 proposes the paradigm of spiking WAMs and builds SpikingWAM on it from these designs.

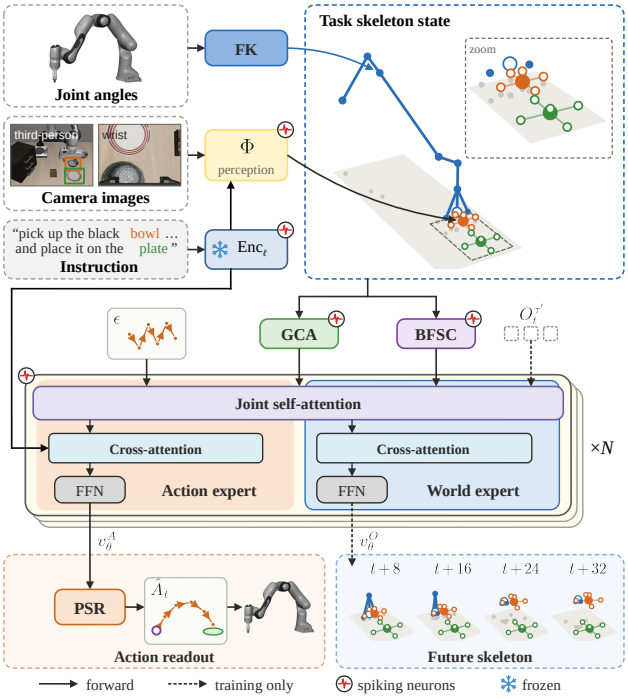


Figure 2. **Architecture of SpikingWAM.** Arm nodes come from forward kinematics, and the spiking front end estimates the object nodes from images. The future skeleton is used only in training.

4.1. Task Skeleton State and Body Frame Spike Coding

The first stage decides what serves as the world state and writes it into spikes. When people manipulate objects, their gaze falls almost only on grasp points and targets [12]. Inspired by this, TSS keeps only the keypoints of the arm and the task objects,

$$o_t = \big[\, \mathrm{FK}(\mathbf{q}_t)\,;\, \Phi(\mathcal{V}_t, \ell) \,\big] \in \mathbb{R}^{n \times 3}, \quad (4)$$

where forward kinematics FK gives the positions of the arm nodes from the joint angles $\mathbf{q}_t$. The perception front end Φ is a small spiking network. It estimates the positions of the task object nodes from the camera images $\mathcal{V}_t$ and the instruction ℓ (Appendix B). Besides object centres, TSS includes face centres that rotate with the object, points on moving parts and the two fingertips. These nodes give the orientation of objects, the state of drawers and doors, and the opening of the gripper. We write $\mathbf{p}_{t,j}$ for the position of node j and ι_j for its descriptor, which records its identity, its role and the object it belongs to (Appendix C). The future world state O_t holds the skeletons of several later frames. TSS thus lets the policy read only the nodes that matter for the task, and its events concentrate on these nodes. Future prediction becomes the generation of node coordinates and no longer needs a video generation model.

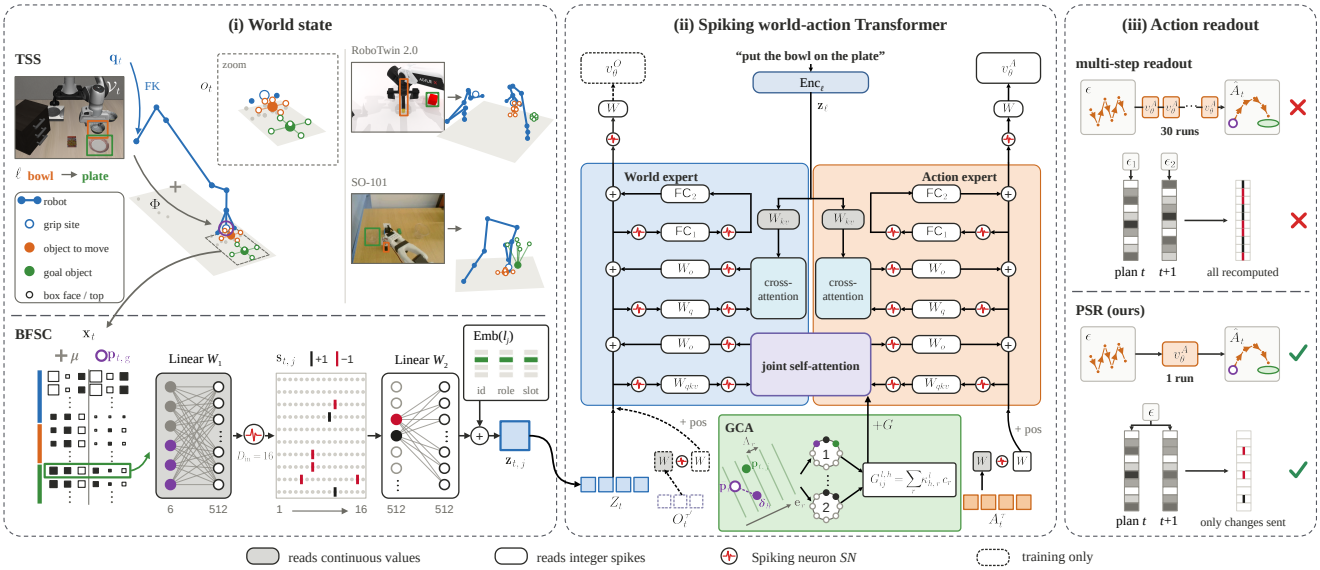


Figure 3. **SpikingWAM.** (i) The perception front end estimates the keypoints of the task objects from camera images. Together with the arm nodes from forward kinematics, they form the current skeleton (TSS), and BFSC writes it into spikes. (ii) The world expert processes the current and future skeletons, and the action expert processes the noisy action chunk. The two experts compute self-attention together in every layer, and GCA adds a relative geometry term to it. (iii) At inference the future skeleton is removed. The readout of Sec. 3.2 runs the network $kN = 30$ times per plan and recomputes everything, while PSR runs it once and passes only changes between plans.

The skeleton contains only coordinates, so the success of manipulation depends on how precisely the coordinates are written into spikes. A coordinate component p enters a neuron of Eq. (1) through the input layer $u = wp + b$. To avoid saturation over a value range R, the layer needs $|w|R/V_{\text{th}} \leq 2D$. The level spacing of a single neuron therefore satisfies

$$\Delta p = \frac{V_{\text{th}}}{|w|} \geq \frac{R}{2D}. \tag{5}$$

With the workspace as the reference and $D = 4$, the level spacing already exceeds 10 cm (Appendix A). Inspired by how the cortex represents targets relative to the arm and the hand [3, 7], BFSC reduces the level spacing in two ways. Each node also receives its coordinates relative to the grip site, which we call relative coordinates. The range R of nodes near the hand thus becomes small. The coordinate input layer uses more levels, $D_{\text{in}} > D$, which raises D,

$$\mathbf{x}_{t,j} = \Big[(\mathbf{p}_{t,j} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma},\ (\mathbf{p}_{t,j} - \mathbf{p}_{t,\text{g}})/\rho\Big], \tag{6}$$

$$\mathbf{z}_{t,j} = W_2\, \mathcal{SN}_{D_{\text{in}}}\big(W_1 \mathbf{x}_{t,j}\big) + \text{Emb}(\iota_j), \tag{7}$$

where $\mathbf{p}_{t,\text{g}}$ is the grip site, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and standard deviation of each axis, ρ is a length scale, and $\text{Emb}(\iota_j)$ embeds the descriptor. The tokens $\mathbf{z}_{t,j}$ of all nodes form Z_t. The output side also shrinks R. Each step $\mathbf{a}$ of a joint angle action is written as its offset $\mathbf{a} - \mathbf{q}_t$ from the current joint angles (Appendix C). BFSC thus reduces the level spacing of Eq. (5), so integer spikes keep the millimetre precision that fine manipulation needs.

4.2. Grid Coincidence Attention

The second stage models the world and actions jointly through self-attention. Actions depend on the relative geometry of hand and object, but Eq. (7) quantizes the coordinates of each node separately. The error of a relative position can therefore reach one level spacing. GCA is inspired by the periodic firing of grid cells [8] and the coincidence detection of cortical neurons [16]. It writes positions as grid spike codes and reads relative geometry directly from the coincidence of codes. A grid spike code consists of several phase rings, and ring r has a direction $\mathbf{e}_r$, a scale Λ_r and P phase neurons,

$$\psi_{r,\nu}(\mathbf{p}) = \mathbb{I}\Big[\big\|\mathbf{e}_r^{\top}\mathbf{p}/\Lambda_r - \nu/P\big\|_{\circ} \leq 1/P\Big], \tag{8}$$

where $\nu = 0, \ldots, P-1$ and $\|\cdot\|_{\circ}$ is the distance to the nearest integer. In head h of layer l, query i uses $\bar{\mathbf{p}}_i$ as its anchor. Its geometry term with node j counts the bits that are active in both codes,

$$G_{ij}^{l,h} = \sum_r \kappa_{h,r}^{l} \big\langle \boldsymbol{\psi}_r(\bar{\mathbf{p}}_i + \boldsymbol{\delta}_h),\ \boldsymbol{\psi}_r(\mathbf{p}_{t,j}) \big\rangle, \tag{9}$$

where $\boldsymbol{\delta}_h$ is a learnable offset and $\kappa_{h,r}^{l}$ is a learnable gain that starts at 0. Tokens of the current skeleton use themselves as anchors, and tokens of the future skeleton and the

action chunk use the grip site. The term $G^{l,h}$ is added to the self-attention scores on the nodes of the current skeleton (Appendix C). The coincidence count is computed by phase directly from the positions given by TSS, so it bypasses the level spacing of Eq. (5). GCA thus recovers the precision that integer spikes lose in relative geometry, and together with BFSC it keeps millimetre precision.

4.3. Persistent Single Step Readout

The third stage reads the action chunk out of its velocity field v_θ^A. With the readout of several samples in Sec. 3.2, the network runs kN times for each plan. It draws new noise every time, so consecutive plans cannot reuse computation (Appendix A). PSR solves the first problem with a one-step readout and the second with a persistent sample stream.

One-step readout. The optimal velocity field of Eq. (3) is $v^*(A_t^\tau, \tau \mid c) = \mathbb{E}[\epsilon - A_t \mid A_t^\tau, c]$. At $\tau = 1$, $A_t^1 = \epsilon$ is independent of A_t, so

$$v^*(\epsilon, 1 \mid c) = \epsilon - \mathbb{E}[A_t \mid c], \tag{10}$$

$$\hat{A}_t = \epsilon - v_\theta^A(\epsilon, 1 \mid c) \approx \mathbb{E}[A_t \mid c]. \tag{11}$$

A single integration step from pure noise thus gives the conditional mean of the action chunk. Training samples noise times that lean toward $\tau = 1$ [2], so v_θ^A is trained best at this end. When the conditional distribution has a single mode, the conditional mean is an executable action. This condition can be checked offline with samples from the model (Appendix D). Otherwise, the same network reads out with several steps as in Sec. 3.2.

Persistent sample stream. The conditional mean in Eq. (11) does not depend on the starting noise. PSR therefore draws the noise only once per episode, and between two consecutive plans only the skeleton in the network input changes. The network uses the incremental firing of sigma-delta networks [24] and delta networks [23]. Each neuron sends only the change of its integer output, and the receiving layer adds it up,

$$\mathbf{I}^l = \mathbf{I}'^l + W^l\big(\mathbf{s}^{l-1} - \mathbf{s}'^{l-1}\big), \tag{12}$$

where primed quantities come from the previous run. Equation (12) equals $W^l\mathbf{s}^{l-1}$ exactly, but the number of accumulations is only proportional to $\|\mathbf{s}^{l-1} - \mathbf{s}'^{l-1}\|_1$ (Appendix E). PSR thus runs the network only once per plan and keeps the savings from sparsity.

4.4. SpikingWAM

To address the three obstacles at the start of this section, we propose a sparse and event-driven paradigm for spiking WAMs. The paradigm builds on the spiking flow matching of Sec. 3.2. At control time t, the instruction ℓ and the world state o_t are first encoded separately. The velocity network v_θ then gives the velocities of the action chunk and the future world state,

$$\mathbf{z}_\ell = \mathrm{Enc}_\ell(\ell), \qquad Z_t = \mathrm{Enc}_o(o_t), \tag{13}$$

$$v_\theta = \Big(v_\theta^A\big(A_t^\tau, \tau \mid Z_t, \mathbf{z}_\ell\big),\ v_\theta^O\big(O_t^{\tau'}, \tau' \mid Z_t, \mathbf{z}_\ell\big)\Big), \tag{14}$$

where Enc_ℓ is a frozen spiking sentence encoder distilled from MiniLM [27, 39] (Appendix C). It outputs the sentence vector $\mathbf{z}_\ell$. The encoder Enc_o writes the world state into spike tokens Z_t. Each of the two velocities has its own loss in the form of Eq. (3), and the future term has weight λ_s. The network has a world expert that processes the world state and an action expert that processes actions [49]. The two experts compute self-attention together in every layer, and the scores on Z_t can include a geometry term $G^{l,h}$. The tokens of the future world state and the action tokens are hidden from each other, and the two share only the tokens of the current world state. Each expert then computes cross-attention to the sentence vector and a feedforward network. The residual stream carries membrane potentials. Each sublayer fires before its weights [44, 45] and is modulated by the noise time through adaLN [25] (Appendix C). Except for adaLN and the grey modules in Fig. 3, every weight layer reads only integer spikes. These layers compute only when a spike fires, and they replace multiply and accumulate with accumulate. Shiftmax [37] normalizes the attention weights with bit shifts (Appendix C). At inference, the future world state is removed. A readout obtains the action chunk from v_θ^A, and the robot executes its first few steps before it replans. Each choice of world state o_t, encoder Enc_o, geometry term $G^{l,h}$ and readout gives a spiking WAM, where the encoder and the geometry term set the geometric precision.

SpikingWAM uses TSS as the world state, BFSC as the encoder, GCA as the geometry term and PSR as the readout,

$$o_t = \mathrm{TSS}(\mathbf{q}_t, \mathcal{V}_t, \ell), \qquad Z_t = \mathrm{BFSC}(o_t), \tag{15}$$

$$G^{l,h} = \mathrm{GCA}(o_t), \qquad \hat{A}_t = \mathrm{PSR}\big(v_\theta^A \mid Z_t, \mathbf{z}_\ell\big), \tag{16}$$

where the operators follow Eq. (4), Eqs. (6) and (7), Eqs. (8) and (9) and Eqs. (11) and (12), respectively. Through the world expert, the loss on the future skeleton shapes the current skeleton features read by the action expert. Future prediction runs only in training and adds no inference energy.

5. Experiments

5.1. Experimental Setup

Benchmarks and energy. LIBERO [21] tests fine manipulation with 40 tasks in four suites, and each task is evaluated for 50 episodes. LIBERO-Plus [5] applies seven kinds

of perturbation to LIBERO in 10,030 variants and tests robustness under perturbations. RoboTwin 2.0 [4] tests bimanual manipulation with 50 tasks. Its Easy setting uses clean scenes, and its Hard setting adds scene randomization. The evaluation protocols are in Appendix F. We estimate the mean energy per plan for a 45 nm process [9], and the perception front end is counted as well. Each accumulate in a layer that reads spikes costs $E_{\mathrm{SOP}} = 0.9\,\mathrm{pJ}$, and each multiply and accumulate in a layer that reads continuous values costs $E_{\mathrm{MAC}} = 4.6\,\mathrm{pJ}$ (Appendix E). For other methods, we estimate one forward pass that generates one action chunk, image encoding included. Their FLOPs come from Spike-driven VLA [38] or from their public architectures.

Model. The perception front end and the policy are trained separately on keypoints from the simulator state, and are connected directly at evaluation (Appendix B). The training images of the front end are randomized procedurally and use no LIBERO-Plus assets. LIBERO and LIBERO-Plus share one model trained on LIBERO demonstrations, which reads out with PSR. Like the compared methods, the RoboTwin 2.0 model uses 50 clean and about 500 randomized demonstrations per task. It reads out with the medoid chunk of $N = 10$ steps and $k = 3$ samples (Sec. 5.3). The architecture and training are in Appendix C.

Compared methods. We divide the compared methods into three groups. The first group contains ANN VLAs and WAMs with more than 1B parameters, such as π_0 [2], $\pi_{0.5}$ [11], Fast-WAM [49] and Cosmos-Policy [15]. The second group contains lightweight ANN VLAs with fewer than 1B parameters, such as TurboVLA [42]. The third group is the spiking VLA Spike-driven VLA [38]. Their results and parameter counts are taken from the cited papers. The main results of SkeleWAM use keypoints from the simulator state as input. The comparison with SpikingWAM in this setting is in Appendix G, and the energy of SkeleWAM is computed on our reproduction (Appendix E).

5.2. Main Results

As shown in Tab. 1, SpikingWAM reaches a mean success of 98.25% on LIBERO, close to the compared SOTA methods Cosmos-Policy [15] and ACoT-VLA [51]. It is 5.85 points higher than Spike-driven VLA [38], which is also a spiking network. It is also 1.35 and 0.65 points higher than $\pi_{0.5}$ [11] and Fast-WAM [49], with only 1/44 and 1/86 of their parameters. SpikingWAM needs only 1.467 mJ per plan, perception front end included (Fig. 1). Under the same rules, SkeleWAM, an ANN world action model counted from the skeleton interface, needs 92.67 mJ. Compared with the policy of SpikingWAM, spiking computation and PSR each save about an order of magnitude (Appendix E). Spike-driven VLA, which reads images, needs 20.9 mJ.

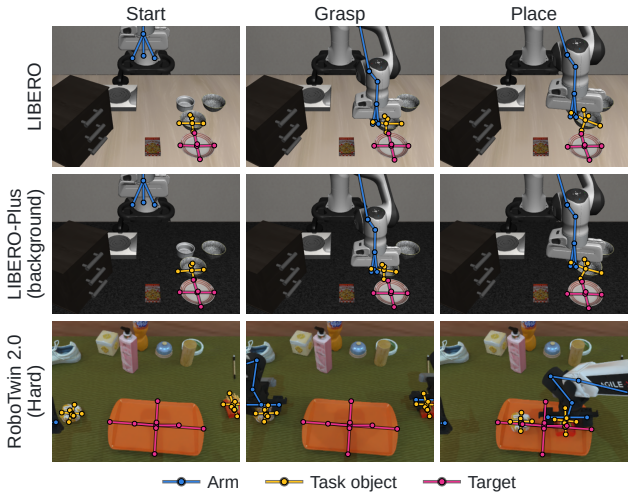


Figure 4. **Closed-loop runs of SpikingWAM.** Each row is one successful episode at the start, the grasp and the last plan before the release, with the task skeleton that the policy reads. Object nodes come from the perception front end, and objects that the instruction does not name have no nodes. The LIBERO-Plus row changes the background of the same task.

As shown in Tab. 1, SpikingWAM reaches a total success of 88.76% on LIBERO-Plus, above all compared ANN and SNN methods. Perturbations of camera, lighting, background and noise change the image, and SpikingWAM estimates only the keypoints of the task objects from the image (Fig. 4). Its success on these four kinds stays between 92.9% and 95.2%, which shows that TSS makes the model focus on the core elements of an embodied scene.

We evaluate SpikingWAM on the 50 bimanual tasks of RoboTwin 2.0. Bimanual manipulation in more complex scenes needs more accurate skeletons, so we enlarge the perception front end to 30M (Appendix B). As shown in Tab. 2, SpikingWAM reaches a mean success of 80.20% with 0.104B parameters. It is 0.45 and 7.38 points higher than $\pi_{0.5}$ [11] and X-VLA [50], with only 1/33 and 1/9 of their parameters. All compared methods use extra pretraining on robot data, while the SpikingWAM policy is trained from scratch only on RoboTwin 2.0 demonstrations. This shows that SpikingWAM, with the task skeleton as its world state, extends to bimanual manipulation and keeps competitive robustness under scene randomization.

5.3. Analysis

This section examines the problems that TSS and BFSC target, and when the one-step readout applies.

Table 1. **Success rates (%) on LIBERO and LIBERO-Plus.** Long is LIBERO-10, and Robot is the robot initial state. Parameters of SpikingWAM include the text encoder and the perception front end.

			LIBERO					LIBERO-Plus							
Method	Venue	Params	Spatial	Object	Goal	Long	Avg.	Camera	Robot	Language	Light	Background	Noise	Layout	Total
ANN, params $\geq$ 1B															
OpenVLA [14]	CoRL'24	7.5B	84.7	88.4	79.2	53.7	76.5	0.8	3.5	23.0	8.1	34.8	15.2	28.5	15.6
π_0 [2]	RSS'25	3.2B	96.8	98.8	95.8	85.2	94.2	13.8	6.0	58.8	85.0	81.4	79.0	68.9	53.6
$\pi_{0.5}$ [11]	CoRL'25	3.4B	98.8	98.2	98.0	92.4	96.9	70.6	50.5	84.4	95.7	93.4	87.4	84.1	79.7
OpenVLA-OFT [13]	RSS'25	7.7B	97.6	98.4	97.9	94.5	97.1	56.4	31.9	79.5	88.7	93.3	75.8	74.2	69.6
Fast-WAM [49]	arXiv'26	6.7B	98.2	**100.0**	97.0	95.2	97.6	16.4	44.5	68.9	78.2	53.7	37.7	60.7	51.5
Cosmos-Policy [15]	ICLR'26	2B	98.1	**100.0**	98.2	**97.6**	**98.5**	75.8	63.3	81.7	96.5	88.9	92.7	82.2	82.2
ACoT-VLA [51]	CVPR'26	3.8B	98.6	99.0	**99.4**	97.0	**98.5**	72.6	82.6	**87.5**	**97.7**	**96.5**	87.8	**88.1**	86.6
GE-Act [18]	ICLR'26	2.5B	–	–	–	–	–	60.7	77.0	77.4	95.8	86.0	90.9	80.2	80.3
ANN, params $<$ 1B															
Evo-1 [19]	CVPR'26	0.77B	92.7	97.7	96.3	92.3	94.8	–	–	–	–	–	–	–	–
TurboVLA [42]	arXiv'26	0.2B	97.0	**100.0**	98.2	95.2	97.6	–	–	–	–	–	–	–	–
X-VLA [50]	ICLR'26	0.9B	98.2	98.6	97.8	**97.6**	98.1	–	–	–	–	–	–	–	–
SNN															
Spike-driven VLA [38]	arXiv'26	0.15B	95.0	95.4	96.6	82.4	92.4	41.2	46.7	48.7	84.2	61.3	34.5	67.5	53.2
SpikingWAM	–	0.077B	**99.0**	**100.0**	98.2	95.8	98.25	**94.6**	**86.5**	76.1	95.2	92.9	**94.3**	84.1	**88.76**

Table 2. **Success rates (%) on RoboTwin 2.0.** Parameters of SpikingWAM include its text encoder and front end.

Method	Venue	Params	Easy	Hard	Avg.
Params $\geq$ 1B					
π_0 [2]	RSS'25	3.2B	65.92	58.40	62.16
$\pi_{0.5}$ [11]	CoRL'25	3.4B	82.74	76.76	79.75
ABot-M0 [43]	arXiv'26	5.9B	86.06	85.08	85.57
GigaWorld-Policy [47]	arXiv'26	12.4B	86.36	85.04	85.70
Qwen-VLA [36]	arXiv'26	–	86.10	**87.20**	86.65
LingBot-VLA [41]	arXiv'26	4.2B	88.56	86.68	87.62
Motus [1]	CVPR'26	8.7B	**88.66**	87.02	**87.84**
Params $<$ 1B					
X-VLA [50]	ICLR'26	0.9B	72.80	72.84	72.82
SpikingWAM	–	0.104B	83.02	77.38	80.20

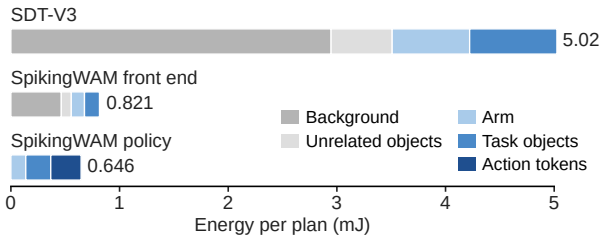


Figure 5. **Energy per plan by region on LIBERO.** The energy of each model is split in proportion to its events in each region (Appendix H). SDT-V3 encodes the two camera images.

Task skeleton. With images as the world state, most events are spent on a background unrelated to the task, and small jitter in the image keeps triggering new events. As shown in Fig. 5, 58.7% of the events of the spiking vision backbone SDT-V3 [46] fall on the background. The policy of SpikingWAM has no events on the background or on unrelated objects. The perception front end only needs to estimate a few keypoints from the images. Its energy per plan is about 1/6 of that of SDT-V3. With one grey level of noise per camera frame, the energy of SDT-V3 rises by 0.64 mJ, while that of the front end rises by only 0.076 mJ. With the front end included, SpikingWAM spends 0.46 mJ per plan on the background, while SDT-V3 spends 2.95 mJ. In the policy, 75% of the attention of action tokens falls on the task objects, above the 57% share of task object nodes (Appendix H). This shows that TSS focuses the policy's events and attention on the parts that matter for the task, which brings out the low power advantage of event-driven SNNs.

Coordinate precision. Encoding coordinates with integer spikes loses precision, and the loss happens in the input layer where coordinates are written into spikes (Appendix A). We read the coordinates back linearly from the spikes of this layer. With BFSC, the error of absolute coordinates comes close to that of an ANN model with the same architecture and continuous input, and the error of relative coordinates is only 3.7 to 4.8 mm. Without BFSC, the error of absolute coordinates grows to 17 to 24 mm. This shows that BFSC recovers the loss from encoding coordinates with integer spikes, so the SNN keeps the millimetre precision that fine manipulation needs.

When the one-step readout applies. The one-step readout gives the conditional mean of the action chunk. This mean is an executable action only when it lies among the model's own samples. We check this condition offline with a ratio. It divides the distance from the mean to its near-

Table 3. **Successes and cost on the real robot.** Each model runs 10 trials. Energy and time are per plan, and time and peak GPU memory are measured in BF16 on the same H100 (Appendix I).

Model	Successes	Energy (mJ)	Time (ms)	Memory (GiB)
X-VLA [50]	5 / 10	3,660	63	3.5
$\pi_{0.5}$ [11]	**8 / 10**	8,740	137	7.1
SpikingWAM	7 / 10	**3.07**	**4.4**	**0.63**

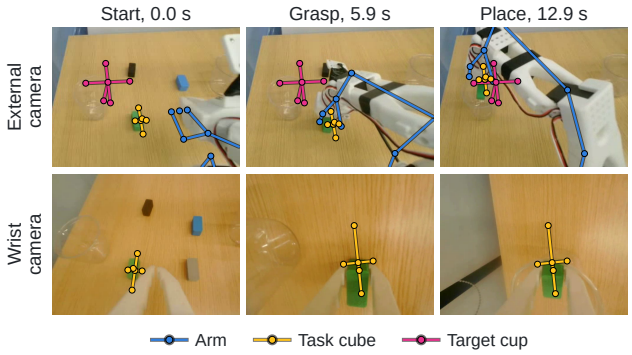


Figure 6. **One successful run of SpikingWAM on the SO-101 arm.** The instruction is "Put the green cube into the left cup." The columns show the start, the grasp and the moment before the cube enters the cup. The top row shows the task skeleton on the external camera, and the bottom row shows the cube nodes that the front end estimates from the wrist camera. The other cubes and the right cup have no nodes.

est sample by the nearest neighbour distance among the samples (Appendix D). The ratio is at most 1.00 on every LIBERO task and at least 2.00 on every RoboTwin 2.0 task, so RoboTwin 2.0 reads out with the medoid chunk of 10 steps. Closed loop results agree with the check. On LIBERO, the one-step readout succeeds more often than the medoid chunk with 10 steps (Sec. 5.5). This shows that the offline check tells whether a benchmark suits the one-step readout without closed loop rollouts. When it does, each plan runs the network only once, which saves time and energy.

5.4. Real Robot Experiments

To test whether SpikingWAM also works in real scenes, we run a pick and place task on an SO-101 arm. The instruction is "Put the green cube into the left cup." The object keypoints are estimated from camera images by spiking networks. SpikingWAM, X-VLA [50] and $\pi_{0.5}$ [11] are trained on the same demonstrations, and each model runs 10 trials (Appendix I). The success rate of SpikingWAM lies between those of the two models, and its energy, inference time and memory are far lower (Tab. 3). SpikingWAM thus performs pick and place efficiently on a real robot (Fig. 6).

Table 4. **Ablation of BFSC and GCA.** Object keypoints come from the perception front end, and the last row is SpikingWAM. Two objects covers the 6 LIBERO-10 tasks that place two objects.

BFSC			Success (%)		
Relative coords.	Input layer	GCA	LIBERO	LIBERO-10	Two objects
✗	✗	✗	91.45	77.4	70.7
✗	✗	✓	92.00	77.4	70.3
✓	✗	✗	95.60	88.2	82.0
✓	✓	✗	97.80	93.6	93.3
✓	✓	✓	**98.25**	**95.8**	**95.0**

Table 5. **Readout and energy.** One LIBERO model with object keypoints from the perception front end. N is the number of integration steps, and the sample mean averages k action chunks. Energy covers only the policy, and the value for $k = 36$ is scaled from the measured breakdown (Appendix D).

				Success (%)	
Readout	N	k	Energy (mJ)	LIBERO	LIBERO-Plus
Medoid	10	3	7.48	91.20	82.93
Sample mean	10	3	7.48	96.15	87.02
Sample mean	10	36	85.85	98.10	–
Single sample	1	1	1.04	97.70	–
PSR	1	1	**0.646**	**98.25**	**88.76**

5.5. Ablation Study

This section ablates BFSC, GCA, future prediction and the readout. As shown in Tab. 4, relative coordinates and the coordinate input layer raise the LIBERO success rate from 91.45% to 97.80%, and the success on the two object tasks from 70.7% to 93.3%. With both in place, GCA raises LIBERO-10 from 93.6% to 95.8% at only 2.18 μJ per plan. Future prediction raises LIBERO-Plus by 1.55 points, mainly under language and robot initial state perturbations (Appendix J). As shown in Tab. 5, the sample mean succeeds more often than the medoid chunk at the same energy, so the conditional mean suits execution better than a single sample. The one-step readout gives the conditional mean directly, so a single sample matches the mean of 36 samples with 10 steps. The persistent sample stream lowers the energy further, and PSR has the highest success rate among all readouts. This shows that all these components are effective. Each raises the success rate or lowers the energy per plan, and together they let an SNN perform fine manipulation at low power.

6. Conclusion

We proposed a sparse and event-driven paradigm for spiking world action models and built on it SpikingWAM, the first spiking world action model. TSS uses only the keypoints of the arm and the task objects as the world state, and BFSC and GCA keep its millimetre precision

in integer spikes. PSR runs the network once per plan and passes only the changes between plans. On simulated benchmarks, SpikingWAM achieves competitive success rates with far less energy than ANN methods, and it also works on a real robot arm. Our results show that the choice of world state is key to spiking WAMs. A task skeleton lets the network spend its events on the arm and the task objects, so its sparsity translates into low energy. We hope this paradigm helps bring WAMs to neuromorphic hardware and advances low power embodied intelligence.

References

[1] Hongzhe Bi, Hengkai Tan, Shenghao Xie, Zeyuan Wang, Shuhe Huang, Haitian Liu, Ruowen Zhao, Yao Feng, Chendong Xiang, Yinze Rong, Hongyan Zhao, Hanyu Liu, Zhizhong Su, Lei Ma, Hang Su, and Jun Zhu. Motus: A unified latent action world model. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. 7

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems*, 2025. 2, 5, 6, 7

[3] Christopher A. Buneo, Murray R. Jarvis, Aaron P. Batista, and Richard A. Andersen. Direct visuomotor transformations for reaching. *Nature*, 416(6881):632–636, 2002. 2, 4

[4] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Zixuan Li, Qiwei Liang, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025. 2, 6

[5] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 38574–38583, 2026. 2, 5

[6] Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. *arXiv preprint arXiv:2505.13447*, 2025. 2

[7] Michael S. A. Graziano, Gregory S. Yap, and Charles G. Gross. Coding of visual space by premotor neurons. *Science*, 266(5187):1054–1057, 1994. 2, 4

[8] Torkel Hafting, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I. Moser. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–806, 2005. 2, 4

[9] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 10–14. IEEE, 2014. 6, 15

[10] Zhixing Hou, Maoxu Gao, Hang Yu, Mengyu Yang, and Chio-In Ieong. SDP: Spiking diffusion policy for robotic manipulation with learnable channel-wise membrane thresholds. *arXiv preprint arXiv:2409.11195*, 2024. 1, 2

[11] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Proceedings of the 9th Conference on Robot Learning*, pages 17–40. PMLR, 2025. 2, 6, 7, 8, 16

[12] Roland S. Johansson, Göran Westling, Anders Bäckström, and J. Randall Flanagan. Eye-hand coordination in object manipulation. *Journal of Neuroscience*, 21(17):6917–6932, 2001. 2, 3

[13] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. In *Proceedings of Robotics: Science and Systems*, 2025. 7

[14] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P. Foster, Pannag R. Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. OpenVLA: An open-source vision-language-action model. In *Proceedings of the 8th Conference on Robot Learning*, pages 2679–2713. PMLR, 2025. 2, 7

[15] Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, et al. Cosmos policy: Fine-tuning video models for visuomotor control and planning. In *International Conference on Learning Representations (ICLR)*, 2026. 1, 2, 6, 7

[16] Peter König, Andreas K. Engel, and Wolf Singer. Integrator or coincidence detector? The role of the cortical neuron revisited. *Trends in Neurosciences*, 19(4):130–137, 1996. 2, 4

[17] Yanjie Li, Sen Yang, Peidong Liu, Shoukui Zhang, Yunxiao Wang, Zhicheng Wang, Wankou Yang, and Shu-Tao Xia. SimCC: A simple coordinate classification perspective for human pose estimation. In *European Conference on Computer Vision (ECCV)*, pages 89–106, 2022. 13

[18] Yue Liao, Pengfei Zhou, Siyuan Huang, Donglin Yang, Shengcong Chen, Yuxin Jiang, Yue Hu, Jingbin Cai, Si Liu, Jianlan Luo, Liliang Chen, Shuicheng Yan, Maoqing Yao, and Guanghui Ren. Genie envisioner: A unified world foundation platform for robotic manipulation. In *International Conference on Learning Representations (ICLR)*, 2026. 1, 2, 7

[19] Tao Lin, Yilei Zhong, Yuxin Du, Jingjing Zhang, Jiting Liu, Yinxinyu Chen, Encheng Gu, Ziyan Liu, Hongyi Cai, Yanwen Zou, Lixing Zou, Zhaoye Zhou, Gen Li, and Bo Zhao. Evo-1: Lightweight vision-language-action model with preserved semantic alignment. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. 7

[20] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023. 3

[21] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, 2023. 2, 5

[22] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023. 3

[23] Daniel Neil, Jun Haeng Lee, Tobi Delbruck, and Shih-Chii Liu. Delta networks for optimized recurrent network computation. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2584–2593, 2017. 5

[24] Peter O'Connor and Max Welling. Sigma delta quantized networks. In *International Conference on Learning Representations*, 2017. 5

[25] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023. 5, 14

[26] Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. *arXiv preprint arXiv:2405.07503*, 2024. 2

[27] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Conference on Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing*, pages 3982–3992, 2019. 5

[28] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019. 1

[29] Juyi Sheng, Hua Wang, and Mengyuan Liu. SkeleWAM: Skeleton world-action modeling for efficient robotic manipulation. *arXiv preprint arXiv:2610.02120*, 2026. 1, 2, 3, 15, 16

[30] Juyi Sheng, Ziyi Wang, Peiming Li, and Mengyuan Liu. MP1: MeanFlow tames policy learning in 1-step for robotic manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026. 2

[31] Ruiqi Song, Dujun Nie, Siyu Teng, Baiyong Ding, Xiaotong Zhang, Dong Li, Chenming Zhang, Yuchen Li, Hangbin Wu, and Long Chen. SpikeVLA: Vision-language-action models with spiking neural networks. *arXiv preprint arXiv:2606.27807*, 2026. 1, 2

[32] 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. *arXiv preprint arXiv:2602.10098*, 2026. 2

[33] Yinqian Sun, Feifei Zhao, Mingyang Lyu, and Yi Zeng. Spiking world model with multicompartment neurons for model-based reinforcement learning. *Proceedings of the National Academy of Sciences*, 122(50):e2513319122, 2025. 1, 2

[34] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017. 13

[35] Qianhao Wang, Yinqian Sun, Enmeng Lu, Qian Zhang, and Yi Zeng. Brain-inspired action generation with spiking transformer diffusion policy model. *arXiv preprint arXiv:2411.09953*, 2024. 1, 2

[36] Qiuyue Wang, Mingsheng Li, Jian Guan, Jinhui Ye, Sicheng Xie, Yitao Liu, Junhao Chen, Zhixuan Liang, Jie Zhang, Xintong Hu, et al. Qwen-VLA: Unifying vision-language-action modeling across tasks, environments, and robot embodiments. *arXiv preprint arXiv:2605.30280*, 2026. 7

[37] Shuai Wang, Malu Zhang, Jingya Wang, Dehao Zhang, Yimeng Shan, Jieyuan Zhang, Yichen Xiao, Honglin Cao, Haonan Zhang, Zeyu Ma, Yang Yang, and Haizhou Li. Bipolar self-attention for spiking transformers. In *Advances in Neural Information Processing Systems*, pages 101586–101611, 2025. 5, 14

[38] Shuai Wang, Malu Zhang, Mingquan Liu, Weihui Dai, Dehao Zhang, Jieyuan Zhang, Yimeng Shan, Zijian Zhou, and Yang Yang. Spike-driven vision-language-action model. *arXiv preprint arXiv:2609.39514*, 2026. 1, 2, 6, 7

[39] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, 2020. 5

[40] Zhendong Wang, Zhaoshuo Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, Ming-Yu Liu, and Yu Zeng. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. *arXiv preprint arXiv:2410.21257*, 2024. 2

[41] Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, Fangjing Wang, Qian Zhu, He Sun, Yong Wang, Shuailei Ma, et al. A pragmatic VLA foundation model. *arXiv preprint arXiv:2601.18692*, 2026. 7

[42] Hengyi Xie, Chenfei Yao, Xianjin Wu, Xuanyang Xi, Yiping Tang, Di Xu, Yingying Zhu, Dingkang Liang, Xiang Bai, and Han Ding. TurboVLA: Real-time vision-language-action model at 32 Hz on an RTX 4090 with $<$1 GB VRAM. *arXiv preprint arXiv:2607.27205*, 2026. 2, 6, 7, 16

[43] Yandan Yang, Shuang Zeng, Tong Lin, Xinyuan Chang, Dekang Qi, Junjin Xiao, Haoyun Liu, Ronghan Chen, Yuzhi Chen, Dongjie Huo, Feng Xiong, Xing Wei, Zhiheng Ma, and Mu Xu. ABot-M0: VLA foundation model for robotic manipulation with action manifold learning. *arXiv preprint arXiv:2602.11236*, 2026. 7

[44] Man Yao, Jiakui Hu, Zhaokun Zhou, Li Yuan, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-driven transformer. *Advances in Neural Information Processing Systems*, 36: 64043–64058, 2023. 2, 5, 14

[45] Man Yao, Jiakui Hu, Tianxiang Hu, Yifan Xu, Zhaokun Zhou, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-driven transformer v2: Meta spiking neural network architecture inspiring the design of next-generation neuromorphic chips. *arXiv preprint arXiv:2404.03663*, 2024. 2, 5, 14

[46] Man Yao, Xuerui Qiu, Tianxiang Hu, Jiakui Hu, Yuhong Chou, Keyu Tian, Jianxing Liao, Luziwei Leng, Bo Xu, and Guoqi Li. Scaling spike-driven transformer with efficient spike firing approximation training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(4):2973–2990, 2025. 1, 2, 3, 7, 16

[47] 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*, 2026. 7

[48] Seonghyeon Ye, Yunhao Ge, Kaiyuan Zheng, Shenyuan Gao, Sihyun Yu, George Kurian, Suneel Indupuru, You Liang Tan, Chuning Zhu, Jiannan Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026. 1, 2

[49] 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. 1, 2, 5, 6, 7

[50] Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Ya-Qin Zhang, Jiangmiao Pang, Jingjing Liu, Tai Wang, and Xianyuan Zhan. X-VLA: Soft-prompted transformer as scalable cross-embodiment vision-language-action model. In *International Conference on Learning Representations (ICLR)*, 2026. 6, 7, 8, 16

[51] Linqing Zhong, Yi Liu, Yifei Wei, Ziyu Xiong, Maoqing Yao, Si Liu, and Guanghui Ren. ACoT-VLA: Action chain-of-thought for vision-language-action models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. 6, 7

[52] Zhaokun Zhou, Yuesheng Zhu, Chao He, Yaowei Wang, Shuicheng Yan, Yonghong Tian, and Li Yuan. Spikformer: When spiking neural network meets transformer. In *International Conference on Learning Representations*, 2023. 2, 12

Spiking World-Action Models for Robotic Manipulation

Supplementary Material

A. Analysis of the Three Obstacles

The designs of Sec. 4 target three parts of the model, namely the world state, geometric precision and the readout. This appendix derives the cost of each part from Eq. (1) and gives measured diagnostics.

Composition of energy. By Eq. (2), each unit spike fired in layer l makes the weight layers that read it perform m_l accumulations. Here m_l is the sum of the output dimensions of these layers. Without reuse across consecutive runs, the energy of one plan is

$$E = kN\Big(E_{\mathrm{SOP}} \sum_{j=1}^{n} \sum_{l} m_l \big\|\mathbf{s}_j^l\big\|_1 + E_{\mathrm{MAC}}\, M\Big), \quad \text{(A.1)}$$

where n is the number of tokens and $\mathbf{s}_j^l$ holds the spikes fired by token j in layer l. M is the number of MACs in the layers that read continuous values during one run, and E_{SOP} and E_{MAC} are the energies of one accumulate and one MAC (Sec. 5.1). In Eq. (A.1), the sum over tokens depends on the world state, and the outer factor kN depends on the readout.

World state. The sum over tokens in Eq. (A.1) depends on how the world state is represented. With images as the world state, n is the number of image patches. Events spread over the image area, and most of the image is background. Small jitter in the image also triggers new events on the patches. For a skeleton, n is only the number of nodes of the arm and the task objects. The events of the policy therefore fall only on these nodes and on the action tokens. Figure 5 and Appendix H give the measurements for both world states.

Geometric precision. The skeleton compresses the scene into coordinates, so the success of manipulation depends on their precision. A coordinate component p enters a neuron of Eq. (1) through the input layer $u = wp + b$. To avoid saturation over a range R, the neuron needs $|w|R/V_{\mathrm{th}} \leq 2D$. The coordinate difference between two adjacent levels therefore satisfies

$$\Delta p = \frac{V_{\mathrm{th}}}{|w|} \;\geq\; \frac{R}{2D}. \quad \text{(A.2)}$$

The three axes of the LIBERO workspace span 1.0 to 1.4 m. With $D = 4$, the level spacing of a single neuron is 12.5 to 17.5 cm. The input layer encodes each coordinate with many neurons. Let S0 be a spiking skeleton model without our designs, with absolute coordinates as input and $D = 4$. Reading coordinates back from the spikes of its input layer still gives an error of 17 to 24 mm. The error is 7.5 to 10.5 mm for the ANN model R0 with continuous input and 9.4 to 9.9 mm for SpikingWAM (Sec. 5.3). S1 adds relative coordinates to S0. On S1, removing the rounding of all inner layers at inference leaves the action error unchanged, at 0.0362 to 0.0378 against 0.0366 with rounding. SpikingWAM gives the same result, so the precision loss happens in the input encoding layer. This loss falls on tasks that need precise placement. S0 reaches 70.7% on the 6 tasks that must place two objects and 95.1% on the other 34 tasks (Tab. 4).

By Eq. (A.2), both a smaller R and a larger D shrink the level spacing, and BFSC does both (Sec. 4.1). The relative position of hand and object is the difference of two separately quantized coordinates, so its error can reach one level spacing. GCA obtains relative geometry directly from the coincidence of their spike codes.

Readout. In Eq. (A.1), the factor kN multiplies the whole energy. With $N = 10$ integration steps and the medoid of $k = 3$ samples, each plan runs the network 30 times, and 95% of the energy is in the action branch (Appendix E). Incremental firing can reuse what stays the same between consecutive integration steps of one sample. However, each plan draws new noise, so the whole input of the action branch changes, and nothing can be reused between consecutive plans. PSR reduces the readout to one step and one sample and then fixes the starting noise, so only changes pass between consecutive plans. Table 5 gives the energy of each readout.

B. Perception Front End

The perception front end Φ of LIBERO (Eq. (4)) estimates the positions of the task object nodes from the third-person and wrist RGB images. The arm nodes still come from forward kinematics.

Architecture. Both images are resized to 160×160 and encoded by one shared spiking convolutional backbone. Convolutions and depthwise separable residual blocks bring the resolution down to 1/16 step by step. Two spiking self-attention layers [52] run on top of it. Their output is upsampled and merged with the 1/8 features, which gives a 20×20 feature map for each view. All layers use $D = 4$ and fire only non-negative spikes. The first convolution reads pixels, and all other layers read only spikes. The front end has

3.46M parameters.

Queries and readout. Each object node has a query vector. It is the sum of the embeddings of the descriptor ι_j and of the object class, plus a linear map of the sentence vector $\mathbf{z}_\ell$, the gripper position and opening, and the camera matrix of the view. The object class comes from the task definition, and the camera matrix holds the known intrinsics and extrinsics. The query takes an inner product with every cell of the feature map. Two spiking fully connected layers then read out three distributions over one axis each. They cover the horizontal pixel coordinate, the vertical pixel coordinate and the height of the node, which is coordinate classification [17]. We take the expectation of each distribution and project the pixel along its camera ray onto the horizontal plane at this height. This gives the position in the robot frame. The positions from the two views are fused with weights given by the softmax of their confidences.

Training. The training images come from rendering all LIBERO demonstrations again with the renderer of the closed loop evaluation. The node positions in the simulator state serve as labels. Besides the original scenes, three more groups of images replace the textures of the table, floor and walls, change the lights and turn the third-person camera. They also move the objects on the table that are not near the gripper at random, which follows domain randomization [34]. All textures are generated procedurally, and no LIBERO-Plus assets are used. During training, half of the third-person images get blur, fog or sensor noise, and instructions are rewritten from a word list. The last 10% of the demonstrations of each task are held out for validation, and the training set has 627k frames. The loss sums three terms. They are the cross entropy of the three distributions against Gaussian soft labels, the L1 errors of the positions from each view and after fusion, and the binary cross entropy of the confidence. We train with AdamW, learning rate 10^{-3} with cosine decay and batch size 128 for 200k steps. The front end is trained alone, and the policy is not fine-tuned on estimated skeletons.

Errors and closed loop results. On the held out demonstrations in the original scenes, the median error of object nodes is 0.28 cm and the 90th percentile is 1.20 cm. Of these nodes, 87.8% lie within 1 cm. The two rows of Tab. B.1 use the same policy and change only the source of the object keypoints. They differ by 0.10 points on LIBERO. On LIBERO-Plus they differ by 1.65 points, and the gap is largest on background, layout and robot initial state.

RoboTwin 2.0. The RoboTwin 2.0 front end reads the head camera and the two wrist cameras. It has the same structure as the front end above, with wider layers. Its training images come from expert demonstrations and from closed loop episodes with the front end, with the node positions in the simulator state as labels. Table B.2 compares front ends of different sizes with the same policy. From 7.4M to 30.1M parameters, the mean success rate rises by 13.2 points, and the energy per plan rises by 7.9 mJ. At 30.1M, the front end and the policy use about the same energy, 20.6 mJ in total. This size gives the highest success rate, so the main text uses it. The front end of the main text has this size and is trained further on images of closed-loop episodes with more backgrounds. With 100 episodes per task, it reaches 83.02% on Easy and 77.38% on Hard (Tab. 2), with 20.7 mJ per plan.

Table B.1. **Closed loop success (%) with object keypoints from the simulator state (§) and from the perception front end.** The policy is the same in both columns.

	Simulator state (§)	**Perception front end**
LIBERO		
Spatial	98.4	99.0
Object	99.8	100.0
Goal	99.0	98.2
LIBERO-10	96.2	95.8
Average	98.35	98.25
LIBERO-Plus		
Camera	96.1	94.6
Robot initial state	88.5	86.5
Language	77.6	76.1
Light	95.3	95.2
Background	95.8	92.9
Noise	95.3	94.3
Layout	86.8	84.1
Total	90.41	88.76

Table B.2. **Perception front ends of different sizes on RoboTwin 2.0.** The policy is the same, and each task runs 10 episodes. Energy is the mean per plan and includes the 10.7 mJ of the policy. The shaded row is the size used in the main text. § marks object keypoints from the simulator state.

Front end	Energy (mJ)	Easy	Hard	Avg.
7.4M	12.7	74.2	59.0	66.6
12.5M	14.7	80.6	69.8	75.2
21.0M	17.5	81.6	73.2	77.4
30.1M	20.6	83.8	75.8	79.8
Simulator state (§)	10.7	89.0	87.8	88.4

C. Model Details

Nodes. On LIBERO, the arm has 11 nodes, namely the origins of Panda links 1 to 7 and of the hand, the grip site and the two fingertips. On RoboTwin 2.0, each of the two arms has 10 nodes, namely the origins of the base link and

of links 1 to 6, the grip site and the two fingertips. Each task object adds a group of nodes. They are its centre, the centres of its top face and four side faces, which rotate with the object, its functional points, and the centres of moving parts such as drawers, doors and knobs. RoboTwin 2.0 also has contact points, target points and orientation points. Coordinates are in the robot frame. On RoboTwin 2.0 they are in the world frame that holds the bases of both arms. The descriptor ι_j is the sum of three embeddings, for the node index, the role and the slot of the object. The same index means the same part in every episode. The future skeleton has one token per frame.

Actions. On LIBERO, the action is the change of the end effector pose plus the gripper command. On RoboTwin 2.0, the action holds the joint angles and grippers of both arms. The joint angles are written as offsets from the current joint angles $\mathbf{q}_t$, and $\mathbf{q}_t$ is added back outside the network. On RoboTwin 2.0, each link node of the arms has one more channel with the measured angle of its joint. The network thus sees the posture to which the action chunk is relative.

Hyperparameters. Spiking neurons use $D = 4$, and the coordinate input layer uses $D_{\text{in}} = 16$. The initial threshold of $\mathcal{SN}_{D_{\text{in}}}$ is scaled by D/D_{in}, so the range stays the same while the number of levels grows. The length scale of relative coordinates is $\rho = 0.2\,\text{m}$, and in bimanual tasks each hand gives its own relative coordinates. The grid spike code has 6 directions and 12 scales, which start at 1 cm and grow by a factor of 1.5. Each ring has 8 phase neurons, so the code has 72 rings. Rings with small scales resolve millimetres, and rings with large scales tell near from far. The offset $\boldsymbol{\delta}_h$ of each GCA head starts from a normal distribution with a standard deviation of 3 cm, and the gain κ starts at 0. In bimanual tasks, each hand takes half of the GCA heads. Shiftmax [37] rounds attention scores to multiples of 1/4 and rounds the denominator to a power of 2.

Layer structure. The residual stream $\mathbf{h}$ carries membrane potentials, and each sublayer fires before its weights [44, 45],

$$\mathbf{h} \leftarrow \mathbf{h} + \boldsymbol{\gamma} \odot \mathcal{F}\big(\mathcal{SN}(\mathbf{u})\big), \qquad \mathbf{u} = (1+\boldsymbol{\beta}) \odot \mathrm{LN}(\mathbf{h}) + \boldsymbol{\alpha}, \tag{C.1}$$

where $\mathcal{F}$ is the sublayer (self-attention, cross-attention or feedforward network) and $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ are modulation coefficients that adaLN [25] computes from the noise time.

Architecture and training. The two benchmarks use the same architecture. They differ in their nodes, their action representation and the amount of training (Tab. C.1). The spiking sentence encoder is distilled from MiniLM, stays frozen during policy training and runs once per episode.

Table C.1. **Architecture and training.**

	LIBERO	RoboTwin 2.0
Experts	world expert and action expert, width 512, 6 layers, 8 heads, feedforward width 3.25 times, block mask	
Parameters	policy 51.3M, spiking text encoder 22.7M, perception front end 3.46M	policy 51.5M, spiking text encoder 22.7M, perception front end 30.1M
Horizon	action chunk of 32 steps with the first 16 executed; future skeleton of 8 frames at 4, 8, …, 32 steps ahead	
Loss	independent noise times for the action and the future skeleton, $\lambda_s = 1$	
Training	60k steps, batch 48, AdamW, learning rate 2×10^{-4} with cosine decay to 2×10^{-5}, EMA 0.999, bf16	150k steps, batch 512, learning rate 6.5×10^{-4} with cosine decay to 2×10^{-5}, others as LIBERO
Data	1,726 demonstrations of the four suites, 279,498 steps	2,500 clean and 24,600 randomized demonstrations, 6,046,558 steps, 1,023,627 instructions seen in training

D. Derivation of the Readout

One-step readout and the sample mean. In Eq. (10), $A_t^1 = \epsilon$ is independent of A_t given c. Conditioning on A_t^1 therefore fixes only ϵ and leaves the distribution of A_t unchanged, so $v^*(\epsilon, 1 \mid c) = \epsilon - \mathbb{E}[A_t \mid c]$. One integration step of size 1 from $\tau = 1$ then gives $\mathbb{E}[A_t \mid c]$. Samples integrated over many steps roughly follow the conditional distribution that the model learns. The mean of k samples therefore tends to the same conditional mean as $k \to \infty$. Training draws the noise time from $\mathrm{Beta}(1.5, 1)$, which leans toward $\tau = 1$.

Offline check of the one-step readout. For each task we take 32 training states. For each state we draw 12 samples of 10 steps with independent noise and one chunk from the one-step readout. We use the arm dimensions of the first 16 executed steps. The ratio divides the distance from a chunk to its nearest sample by the median distance from each sample to its nearest neighbour (Tab. D.1). When the samples form one cluster, the sample mean gives a ratio of about 0.7. On LIBERO, the gripper samples split into two groups by the closing time, and the one-step readout gives a gripper ratio of 7.84. Execution uses only the sign of the gripper command, so this does not affect it.

Table D.1. **Offline check of the one-step readout.** Mean over tasks, with the range in brackets.

	LIBERO (40 tasks)	RoboTwin 2.0 (50 tasks)
Mean of 12 samples	0.67 (0.63–0.70)	1.05 (0.73–2.95)
One-step readout	0.88 (0.81–1.00)	3.10 (2.00–5.07)

Paired episodes of the readouts. The readouts in Tab. 5 are compared episode by episode (notation in Appendix F). The sample mean with 10 steps against the medoid chunk ($k = 3$) gives 155 : 56. The one-step single sample against the mean of 36 samples with 10 steps gives 30 : 38. PSR against a one-step single sample with new noise in each plan gives 41 : 30. PSR against the medoid chunk gives 170 : 29 on LIBERO and 1,001 : 416 on LIBERO-Plus.

E. Energy Counting

Energy model and counting rules. Each accumulate in a layer that reads spikes costs $E_{\mathrm{SOP}} = 0.9\,\mathrm{pJ}$, and each MAC in a layer that reads continuous values costs $E_{\mathrm{MAC}} = 4.6\,\mathrm{pJ}$, for a 45 nm process [9]. Weight layers that read only spikes are counted by the accumulates that unit spikes trigger. Layers that read continuous values, layer normalization and adaLN modulation are counted as MACs. The policy and the perception front end both run with incremental firing (Eq. (12)), so a layer accumulates only the weight columns of the spikes that changed. The first convolution of the front end reads pixels and is counted as MACs. GCA follows the same rule. Each query spike performs one AND with each key, and each coincidence adds one gain in each layer. Each of these counts as one accumulate, and together they cost 2.18 μJ per plan. Energy is the mean per plan. On LIBERO it is averaged over 446 plans on the demonstration trajectories of the 40 tasks, with object keypoints from the perception front end. On RoboTwin 2.0, the policy is averaged over 708 plans on the demonstration trajectories of the 50 tasks, and the perception front end over 2,150 plans of closed-loop episodes. The instruction is encoded only once per episode. The spiking MiniLM needs 0.578 mJ, and the text keys and values of the action expert need another 21.7 μJ. This part is not counted in the energy per plan.

Table E.1. **Energy per plan of the final LIBERO policy.** Accumulates (SOP) and MACs are counted in millions.

	Skeleton		Action		
Counting	SOP	MAC	SOP	MAC	Energy (mJ)
Full computation	781.7	2.12	738.4	2.70	1.390
Skipping unchanged tokens	731.2	2.12	687.2	2.47	1.298
Incremental firing	388.0	2.04	305.3	2.39	0.644
Incremental firing + GCA	adds 2.42M accumulates				**0.646**

Skipping the tokens whose input is unchanged is the counting rule for ANN models, and SkeleWAM is counted with this rule.

Source of the energy gap. The original SkeleWAM paper [29] does not release its node definition or its text encoder. We reproduce SkeleWAM with the architecture of the original paper, which is R0 in Appendix A. The reproduction uses our node definition and frozen MiniLM token features, and it needs 92.67 mJ per plan. On one SpikingWAM model, this gap of about 144 times can be measured step by step. With the same medoid readout at $k = 3$ and $N = 10$, spiking computation and incremental firing lower the energy from 92.67 mJ to 7.48 mJ (12.4 times). The one-step readout with a single sample lowers it further to 1.04 mJ (7.2 times), and the persistent sample stream lowers it to 0.646 mJ (1.6 times). Under the medoid readout, the skeleton branch needs 0.359 mJ and the action branch 7.12 mJ. The final RoboTwin 2.0 policy reads out with the medoid chunk of 10 steps. It needs 10.7 mJ per plan, and 90.2 mJ with full computation.

F. Evaluation Protocol

Each of the four LIBERO suites has 10 tasks. Each task runs one episode on each of its 50 official initial states, which gives 2,000 episodes. The seven perturbations of LIBERO-Plus are camera, robot initial state, language, lighting, background, sensor noise and object layout, and each variant runs one episode. RoboTwin 2.0 runs 100 seeds per task on both Easy and Hard, and none of the evaluation instructions appear in training. The policy uses the EMA weights at the end of training.

Paired episodes. Two models are compared episode by episode on the same initial states or seeds. We write the result as $a : b$, where a counts the episodes in which only the first model succeeds and b those in which only the second succeeds.

G. Comparison with SkeleWAM

The main results of SkeleWAM [29] take object keypoints from the simulator state (Sec. 5.1). We therefore compare it with SpikingWAM §, whose object keypoints also come from the simulator state (Tab. G.1). The two have similar parameter counts. SpikingWAM § is 2.71 points higher in total. The gain comes mainly from robot initial state and layout, by 13.3 and 17.8 points, and SpikingWAM § is 11.4 points lower on language. With the perception front end, SpikingWAM reaches 88.76% on LIBERO-Plus (Tab. 1), still above SkeleWAM §.

H. Statistics of Events and Attention

We take the first demonstration of each of the 40 LIBERO tasks and keep one frame every 16 steps, which gives 446 frames. Both cameras are rendered at 256×256 together with instance segmentation. Pixels are split into four regions, namely background (table, floor and walls), arm, task objects (the objects of interest in the BDDL file) and other

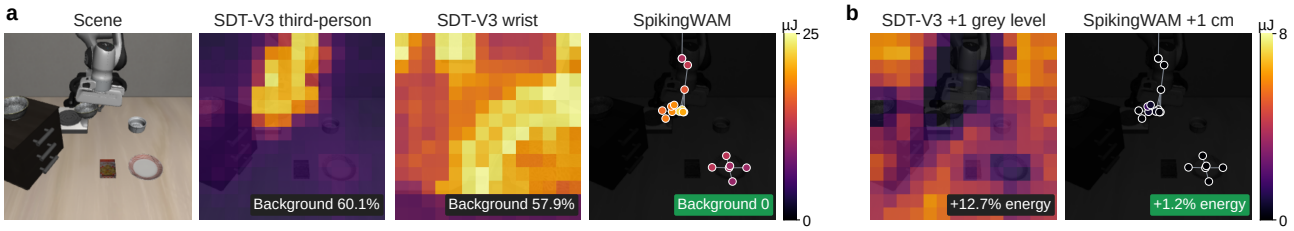


Figure H.1. **Where events fall.** (a) Incremental events of one plan. The cells of SDT-V3 and the nodes of SpikingWAM share one colour scale. Labels give the share of events on the background over all 446 frames. Action tokens have no spatial position and are not drawn. (b) Events added by input noise of 1 grey level per camera frame or 1 cm per object keypoint, shown on the third-person camera. Labels give the energy increase.

Table G.1. **Success rates (%) on LIBERO-Plus with object keypoints from the simulator state.** SkeleWAM results come from its paper. Both policies have 0.051B parameters.

	SkeleWAM § [29]	**SpikingWAM §**
Camera	**96.1**	**96.1**
Robot initial state	75.2	**88.5**
Language	**89.0**	77.6
Light	**96.6**	95.3
Background	**96.7**	95.8
Noise	**95.4**	95.3
Layout	69.0	**86.8**
Total	87.7	**90.41**

objects. By pixel area, the background covers 64.3%, the arm 13.6%, the task objects 12.6% and the other objects 9.6%. SDT-V3 [46] uses its official ImageNet weights, with both images resized to 224×224. The events at each position are split among the regions by the pixel shares inside its cell. For pixel noise, each camera frame gets independent Gaussian noise with standard deviation $s/255$ before resizing.

Between consecutive plans, incremental firing keeps 48.1% of the full events for SDT-V3 and 49.6% for the skeleton branch of the SpikingWAM policy. The two inputs therefore differ mainly in where the events fall (Fig. H.1).

The events of the perception front end are split among the regions in the same way. The keypoint queries and the readout have no image position, so their 0.7% of the events is not assigned to any region in Fig. 5.

Attention. On the same 446 frames, with keypoints estimated by the perception front end, 75% of the attention that the action tokens of SpikingWAM pay to nodes falls on the task objects. Task object nodes make up only 57% of all nodes, so the attention concentrates on the task objects. For the image VLA TurboVLA [42], we use its public weights trained on LIBERO. The attention of its action head follows the area of each region. The background receives 0.62 of this attention, close to its 0.64 share of the area.

I. Real Robot Experiments

Platform and task. The experiments run on an SO-101 arm. The table holds four cubes of different colours and two transparent cups, and the instruction is "Put the green cube into the left cup." All three models use the same 324 teleoperated demonstrations. X-VLA [50] and $\pi_{0.5}$ [11] are fine-tuned from their public weights. SpikingWAM is trained from scratch with the same policy architecture as in simulation, and it reads out with the medoid chunk of 5 steps ($k = 3$).

Keypoints. The perception front end (0.81M parameters) estimates the cube nodes at every plan. A fine-tuned SDT-V3 [46] estimates the cup nodes once at the start of the episode. The label of a cup node is the gripper position at release in the demonstrations. On 18 held out demonstrations, the median error of the cube nodes at the grasp time is 0.32 cm.

Measuring cost. The energy of SpikingWAM is counted with the rules of Appendix E over all plans of the 10 trials, including the perception front end. The energy of locating the cups and of reading the scene at the start of an episode is spread over the plans. The energy of X-VLA and $\pi_{0.5}$ is estimated with the rules of Sec. 5.1, and image encoding counts the image streams that each model actually processes. All three models run on the same H100 in BF16 with batch size 1, and X-VLA and $\pi_{0.5}$ use their official inference code. Inference time is the mean time of one inference over 18 validation demonstrations, including preprocessing and excluding network transfer. Peak memory covers loading and all inference.

J. Role of Future Prediction

The model that learns only actions removes the future skeleton tokens and their loss, and all other settings match SpikingWAM. Both read out with PSR and generate no future skeleton at inference. Table J.1 gives the results.

The two are on par on LIBERO. SpikingWAM against the model that learns only actions gives $36:31$ paired episodes on LIBERO and $607:451$ on LIBERO-Plus (notation in Appendix F). The gap concentrates on the language and robot initial state perturbations, and the other five categories differ by at most 1.5 points. On language, the gap concentrates on LIBERO-Spatial (85.6 against 74.4) and LIBERO-Goal (66.8 against 58.0). Both suites need the instruction to pick out the target within the same scene.

Table J.1. **Removing future prediction (%).** Object keypoints come from the perception front end, and both models read out with PSR.

	Actions only	**SpikingWAM**
LIBERO	98.00	**98.25**
LIBERO-Plus	87.21	**88.76**
Camera	94.4	**94.6**
Robot initial state	83.3	**86.5**
Language	69.7	**76.1**
Light	93.7	**95.2**
Background	**93.8**	92.9
Noise	94.0	**94.3**
Layout	**84.7**	84.1