

Supplementary Information

Efficient visual motion prediction using memristive spatiotemporal compression

Authors

Ao Xu¹, Lun Lu¹, Yinan Wang^{1*}, Youpeng Wu¹, Lang Qin¹, Mingxin Deng¹, Chang Liu¹, Changlin Chen¹, Xiao Fu³, Hangyu Xu³, Binhan Du⁴, Hui Xu¹, Qi Liu^{2*}, Qingjiang Li^{1*}

Affiliations

¹College of Electronic Science and Technology, National University of Defense Technology, Changsha 410073, China

²State Key Laboratory of Integrated Chips and Systems, College of Integrated Circuits and Micro-Nano Electronics, Frontier Institute of Chip and System, Fudan University, Shanghai, China

³State Key Laboratory of Infrared Physics, Shanghai Institute of Technical Physics, Chinese Academy of Sciences, Shanghai 200083, China

⁴College of Intelligence Science and Technology, National University of Defense Technology, Changsha, 410073, China

Correspondence and requests for materials should be addressed to Y.Wang, Q.Liu, Q.Li
(Email: wangyinan@nudt.edu.cn, qi_liu@fudan.edu.cn, qingjiangli@nudt.edu.cn)

Table of Contents

23	<i>Supplementary Notes</i>	4
24	Supplementary Note 1 Device electrical characterization	4
25	Supplementary Note 2 Hardware implementation of frame-difference-driven motion-	
26	state accumulation	4
27	Supplementary Note 3 In situ spatiotemporal encoding of dynamic visual scenes.....	5
28	Supplementary Note 4 Comparison between transient event encoding and memristive	
29	motion-state accumulation	6
30	Supplementary Note 5 Generalization of dynamic visual encoding across diverse scenes	7
31	Supplementary Note 6 Unified hardware deployment for static recognition and dynamic	
32	perception.....	7
33	Supplementary Note 7 FPGA temporal-difference implementation.....	9
34	Supplementary Note 8 Array mapping and current-domain execution of the MLP	
35	predictor	10
36	Supplementary Note 9 CNN-LSTM baseline for single-object prediction	11
37	Supplementary Note 10 Hardware implementation of dual-object trajectory prediction	
38		12
39	Supplementary Note 11 Candidate-conditioned current readout for resolving dual-object	
40	ambiguity	12
41	Supplementary Note 12 CNN-LSTM baseline for dual-object trajectory prediction	13
42	Supplementary Note 13 Memristive mixed-signal hardware platform	13
43	Supplementary Note 14 Closed-loop resistance targeting by adaptive pulse	
44	programming.....	14
45	Supplementary Note 15 Latency and energy comparison for single-object tracking and	
46	trajectory prediction	14
47	Supplementary Note 16 Latency and energy comparison for dual-object tracking and	
48	trajectory prediction	17
49	<i>Supplementary Figures</i>	20
50	Fig S1. Device electrical characterization.....	20
51	Fig S2. Experimental motion-state accumulation in a memristor array.	21
52	Fig S3. Dynamic visual encoding in a memristor array.....	22
53	Fig S4. Comparison between event-based encoding and memristive motion-state	
54	accumulation.	23

55	Fig S5. Generalization of memristive dynamic visual encoding across diverse driving	
56	scenes.	24
57	Fig S6. Unified hardware implementation for static recognition and dynamic perception.	
58		25
59	Fig S7. FPGA implementation of parallel temporal-difference encoding.	26
60	Fig S8. Hardware mapping and current-domain validation of the motion-prediction	
61	MLP.....	27
62	Fig S9. Architecture and performance of the raw-frame CNN–LSTM prediction	
63	baseline.	28
64	Fig S10. Hardware implementation of dual-object trajectory prediction.	29
65	Fig S11. Candidate-conditioned current readout for dual-object coordinate association.	
66		30
67	Fig S12. CNN–LSTM baseline for multi-step trajectory prediction.	31
68	Fig S13. Memristive mixed-signal hardware platform.....	32
69	Fig S14. Closed-loop target-resistance programming of memristive devices.	33
70	<i>Supplementary Tables</i>	<i>34</i>
71	Supplementary Table 1 System-level latency and energy comparison for single-object	
72	trajectory prediction.	34
73	Supplementary Table 2 System-level latency and energy comparison for dual-object	
74	trajectory prediction.	34
75	<i>Supplementary Videos</i>	<i>35</i>
76	Supplementary Video 1 Hardware future-motion prediction for a single moving object	
77		35
78	Supplementary Video 2 Hardware multi-object trajectory prediction and proximity-risk	
79	assessment	35
80		
81		
82		

Supplementary Note 1 Device electrical characterization

To evaluate the bipolar switching behaviour, consecutive hysteresis measurements were performed by sweeping the write voltage from 0 to +1.5 V, back to 0 V, then to -2.0 V and finally back to 0 V, with a voltage increment of 0.05 V. Positive and negative voltage sweeps were applied using gate voltages of 1.6 V and 3.0 V, respectively. The write pulse width was 3 μ s, and the device state was subsequently read using a 0.2 V read voltage with a 1 μ s read pulse. A total of 50 consecutive cycles were recorded. As shown in Fig. S1a, the device exhibited reproducible bipolar hysteresis, with SET occurring during the positive-voltage sweep and RESET during the negative-voltage sweep. The averaged traces retained a clear separation between the forward and reverse sweep branches, while the individual cycles remained within a consistent switching envelope.

As shown in Fig. S1b, the device exhibited continuously tunable and distinguishable $I - V$ characteristics across 80 programmed conductance states. An initial RESET pulse (-2.0 V, 3 μ s) was followed by progressively increased SET pulses (0.55-1.05 V, 1-3 μ s). The $I - V$ characteristics were measured from 0 to 0.2 V at a gate voltage of 3.0 V using a 1 μ s read pulse, demonstrating stable analogue conductance modulation suitable for multilevel memory and neuromorphic computing applications. As shown in Fig. S1c, eight target conductance states ranging from 20 to 250 μ S were programmed by closed-loop retargeting. Each state was subsequently read ten times at a read voltage of 0.2 V and a gate voltage of 2.0 V, with no additional programming pulses applied during the retention measurement. The conductance distributions remained distinguishable across all eight states, demonstrating reliable retention of multilevel analogue conductance states.

As shown in Fig. S1d, following an initial RESET pulse, 32 selected devices were subjected to five consecutive SET/RESET cycles. SET pulses were swept from 0.5 to 1.5 V at a gate voltage of 1.6 V, whereas RESET pulses were swept from -0.6 to -1.5 V at a gate voltage of 3.3 V, both used a pulse width of 10 μ s. Conductance was read at 0.2 V with a gate voltage of 2.0 V after each programming pulse. Thin lines represent individual devices, solid lines indicate the median, and shaded regions denote the interquartile range. The devices showed reproducible potentiation and depression characteristics over repeated cycles.

Supplementary Note 2 Hardware implementation of frame-difference-driven motion-state accumulation

To demonstrate in situ updating and history retention of consecutive visual changes in the memristor array, an 8×16 subarray was selected from the 64×128 array and driven by a six-

frame sequence containing a horizontally moving target. The target was represented by a 3×3 grayscale spot whose centre moved progressively from the left to the right of the field of view.

The frame-to-frame pixel difference was linearly converted into a desired conductance change. A positive difference above the threshold generated a SET operation, a negative difference below the negative threshold generated a RESET operation, and variations within the threshold window produced no update. The threshold was set to 5% of the available conductance range, corresponding to $6.5 \mu S$. The write strength was normalized according to the magnitude of the valid difference and mapped to SET pulses of up to $+2.5 V$ or RESET pulses down to $-2.4 V$. A gate voltage of $2.0 V$ and a write-pulse width of $3 \mu s$ were used. Each device was read immediately after programming, and the measured conductance became the initial state for processing the next frame, without resetting the array between frames.

As shown in Fig. S2, the first row shows the consecutive input frames, and the second row presents the corresponding write-voltage maps. Newly occupied target positions generate positive SET voltages, whereas previously occupied positions generate negative RESET voltages. The third row shows the measured conductance map after each update. As the target moves, the conductance distribution evolves along the motion direction while retaining the effects of preceding differential updates, thereby forming a distributed state that represents the recent motion path. These results demonstrate that consecutive frame changes can be consolidated directly into the instantaneous conductance map, providing a hardware state with temporal history for subsequent row-column current projection and future-position prediction.

Supplementary Note 3 In situ spatiotemporal encoding of dynamic visual scenes

To evaluate the capability of the memristor array to process real dynamic visual scenes, a traffic video was resized and mapped onto a 64×114 array. Consecutive grayscale frames were subjected to temporal differencing and threshold comparison. Positive and negative intensity changes triggered SET and RESET operations, respectively, whereas static-background variations below the threshold produced no update. The array was not globally reset between frames, and the measured conductance after each update served as the initial state for the subsequent frame.

Fig. S3 presents representative results at frames 0, 5, 9 and 14. The input scene contains a pedestrian crossing the road and moving vehicles. As successive frames are applied, local conductance updates induced by the moving objects are accumulated in situ within the array. Consequently, the instantaneous conductance distribution reflects both the current object location and spatial traces generated by its preceding motion. The full conductance map, G_t , therefore acts as a dynamic visual memory containing recent trajectory history.

To extract instantaneous spatial variation at the current time, the frame-to-frame differential conductance was calculated as $\Delta G = G_t - G_{t-1}$. Positive and negative conductance changes indicate newly occupied and vacated regions, respectively, thereby highlighting the moving pedestrian and vehicles against the static background. Complementary to the motion-history information stored in the accumulated conductance map, ΔG_t provides the current positional and boundary changes. Together, these results show that the memristor array simultaneously encodes motion history and instantaneous spatial information during the same physical-state update process, providing inputs for subsequent row – column current projection and future trajectory prediction.

Supplementary Note 4 Comparison between transient event encoding and memristive motion-state accumulation

To compare conventional event representation with the proposed memristive dynamic-vision encoding, the same consecutive image frames were subjected to temporal differencing and threshold comparison. In event-camera-like processing, positive and negative intensity changes generate ON and OFF events, respectively, forming a two-dimensional event map at the current time step. This representation accurately highlights instantaneous motion boundaries, but each event map primarily describes local changes between adjacent frames. Exploiting motion history over a longer temporal interval therefore requires subsequent buffering or integration of the event sequence.

In the proposed system, the same positive and negative temporal contrasts are converted into SET and RESET voltages and applied to the corresponding 1T1R cells. Because each update acts on the device state retained from the preceding time step, the array conductance G_t evolves progressively with the input sequence without global resetting between frames. The instantaneous conductance map therefore represents not only the current object location but also state changes induced by preceding motion, providing single-state consolidation of the recent trajectory.

[Fig. S4](#) compares the results for two adjacent frames. Both the event map and the frame-to-frame differential conductance, ΔG_t , highlight instantaneous spatial changes and exhibit corresponding positive and negative polarities. In contrast, the complete conductance map G_t additionally retains motion traces established by previous updates. The memristor array thus provides instantaneous variation and accumulated history within the same physical state, ΔG_t represents the current temporal contrast, whereas G_t stores the recent motion state for subsequent row-column current projection and trajectory prediction without explicit buffering of the event sequence.

Supplementary Note 5 Generalization of dynamic visual encoding across diverse scenes

To evaluate the applicability of the memristive dynamic-vision encoding method under different environments and traffic conditions, four video sequences with varying background complexity, illumination, visibility and object distributions were mapped onto a 64×114 memristor array. The scenes include a snow-covered urban road, a low-visibility roadway, a high-contrast open environment and a multi-vehicle intersection. All sequences were processed using the same frame-differencing, thresholding and in situ SET/RESET update procedure.

[Fig. S5](#) shows a representative input frame, the accumulated conductance map and the frame-to-frame differential conductance for each scene. Despite substantial differences in background texture, image contrast and moving-object scale, localized conductance updates were consistently generated by the consecutive visual inputs. The accumulated state G_t retained motion traces of the vehicles over recent frames, whereas the differential conductance, ΔG , highlighted newly occupied and vacated regions at the current time step. These results indicate that the encoding mechanism is not restricted to a specific background or fixed object appearance and can simultaneously extract instantaneous motion information and retain recent motion history under diverse real-world traffic conditions.

Supplementary Note 6 Unified hardware deployment for static recognition and dynamic perception

To examine whether the sensory-memory front end can support both static recognition and dynamic perception within a shared hardware pathway, we constructed a unified static–dynamic network using Moving MNIST¹ and deployed it on the memristive mixed-signal platform ([Fig. S6a](#)). Two adjacent frames were used as inputs. The input images were encoded into the first-layer memristor array through threshold-gated single-pulse SET/RESET operations, allowing the conductance state to evolve with successive observations. The first convolution operated directly on this conductance-coded sensory state, and the rectified difference between the first-layer responses at two consecutive time steps was propagated through the subsequent convolutional and fully connected layers. Static content and temporal variation were therefore processed through the same hardware inference path without introducing an additional temporal-processing module.

We first evaluated static-recognition fidelity by comparing the memristive hardware implementation with the corresponding software baseline ([Fig. S6b,c](#)). For each condition, 2,000 test images were randomly sampled and the evaluation was repeated five times. Under noise-free inputs, the software baseline and memristive hardware achieved accuracies of $90.40 \pm 0.34\%$ and $90.15 \pm 0.38\%$, respectively, indicating negligible accuracy loss after hardware deployment. When

input noise was introduced, the software accuracy decreased to $27.47 \pm 0.73\%$, whereas the memristive implementation retained $73.25 \pm 0.51\%$. These results show that the sensory-memory hardware preserves static-recognition accuracy under clean inputs while providing substantially improved robustness to input perturbations.

Dynamic perception was evaluated using two complementary tasks: instantaneous event extraction and accumulated motion-trail reconstruction. The instantaneous event map describes newly occupied and vacated regions between adjacent observations, whereas the accumulated trail represents motion information retained over the complete sequence.

The proposed sensory-memory approach was compared with frame differencing, pseudo-event generation and a compact Tiny RNN under identical input sequences and evaluation criteria (Fig. S6d,e). Frame differencing and pseudo-event generation served as non-learning digital baselines, whereas Tiny RNN provided a trainable temporal baseline operating on the image sequence. Under noise-free conditions, the proposed method achieved the strongest combined IoU and F1 performance for instantaneous event extraction. For motion-trail reconstruction, Tiny RNN achieved the highest overall performance, while the proposed method remained competitive without task-specific temporal training and clearly outperformed the two non-learning baselines.

When Gaussian noise with a standard deviation of 0.2 was added to the inputs, the proposed sensory-memory approach retained the highest IoU and F1 scores for both event extraction and motion-trail reconstruction (Fig. S6e). Frame differencing and pseudo-event generation converted input fluctuations directly into spurious temporal responses, while Tiny RNN also degraded under the noisy input distribution. In contrast, threshold gating suppresses small random fluctuations, whereas physical conductance accumulation retains persistent motion-induced changes.

The sequence-level processing costs were finally compared using identical input and output boundaries (Fig. S6f,g). The proposed sensory-memory approach requires $224 \mu s$ per sequence, compared with $440 \mu s$ for frame differencing, $730 \mu s$ for pseudo-event accumulation and $800 \mu s$ for Tiny RNN, corresponding to approximately $2.0\times$, $3.3\times$ and $3.6\times$ speedups, respectively. The corresponding energy consumption is $202 \mu J$ per sequence, compared with approximately $52 mJ$, $66 mJ$ and $20 J$ for the three baselines, yielding energy reductions of approximately 257-fold, 327-fold and 10^5 -fold. These results show that encoding and accumulating temporal information directly in the sensory-memory state substantially reduces the latency and energy costs of explicit digital temporal processing while supporting static and dynamic visual tasks within a shared hardware pathway.

Supplementary Note 7 FPGA temporal-difference implementation

Temporal differencing was implemented on the FPGA as a streaming pipeline with 64-pixel parallelism (Fig. S7a). For an $m \times n$ 8-bit grayscale image, each row is partitioned into blocks of up to 64 adjacent pixels, which are transferred and processed in parallel. The datapath comprises an input interface, two ping-pong frame buffers, 64 parallel subtractor–comparator units, SET/RESET encoding logic and a row-mask accumulator. A lane-valid mask disables unused channels in the final block when the image width is not divisible by 64.

The two frame buffers provide spatially corresponding pixels from consecutive frames. During processing of frame t , one buffer stores the incoming pixels while the other supplies pixels from frame $t-1$; their roles are exchanged at each frame boundary. The first frame initializes the reference buffer and generates no programming commands, whereas each subsequent frame is compared with its immediate predecessor to generate SET and RESET decisions.

The block-level dataflow is detailed in Fig. S7b. For each valid input block, 64 current-frame pixels $P_{i,j}(t)$ and 64 co-located previous-frame pixels $P_{i,j}(t-1)$ are read and processed concurrently. The 64 parallel subtractors calculate the signed temporal differences

$$\Delta P_{i,j}(t) = P_{i,j}(t) - P_{i,j}(t-1) \quad (1)$$

Each temporal difference is subsequently compared with a programmable threshold θ :

$$\begin{cases} \Delta P_{ij} > \theta \rightarrow SET \\ \Delta P_{ij} < -\theta \rightarrow RESET \\ |\Delta P_{ij}| \leq \theta \rightarrow No\ update \end{cases} \quad (2)$$

A positive temporal contrast therefore produces a SET decision, whereas a negative temporal contrast produces a RESET decision. Pixel variations within the threshold interval are suppressed to reduce responses to small background fluctuations and input noise.

The 64 pixel-wise decisions generated in each clock cycle are encoded as separate 64-bit SET and RESET masks, as shown in Fig. S7b. The l -th bit of each mask corresponds to the l -th pixel lane in the current input block. Thus, $SET\ mask[l] = 1$ indicates that the corresponding pixel requires a SET operation, whereas $RESET\ mask[l] = 1$ indicates that it requires a RESET operation. When both bits are zero, no update is generated for that pixel. The SET and RESET decisions are mutually exclusive.

The row-level accumulator concatenates the masks from the $B = \lceil n/64 \rceil$ blocks belonging to the same image row, producing an n -bit SET row mask and an n -bit RESET row mask, as shown in Fig. S7a,b. These row masks preserve the original column positions of all detected events and allow the downstream array controller to receive vectorized row commands rather than a large sequence of individual pixel commands.

As shown in Fig. S7c, the output-valid signals indicate completion at different levels of the

pipeline. The *block_valid* signal indicates that the SET and RESET masks for the current 64-pixel block are valid. The *frame_done* signal is asserted together with the final valid block of the final row, indicating that all temporal-difference decisions for the current frame have been generated. After the fixed pipeline-filling interval, the architecture sustains a throughput of 64 pixels per clock cycle. The frame-level decision latency can be expressed as:

$$T_{frame} = \frac{m \left\lceil \frac{n}{64} \right\rceil + N_{pipe}}{f_{clk}} \quad (3)$$

where N_{pipe} is the fixed pipeline depth and f_{clk} is the FPGA operating frequency. The FPGA therefore converts consecutive grayscale frames directly into spatially aligned SET and RESET programming masks for the sensory array, without constructing an intermediate event frame.

Supplementary Note 8 Array mapping and current-domain execution of the MLP predictor

The multilayer perceptron used for motion prediction comprises a 178-dimensional input layer, a 32-neuron hidden layer and a two-neuron output layer (Fig. S8a). The input vector is formed by concatenating 64 row-projection features and 114 column-projection features, whereas the two outputs represent the predicted target displacement along the two spatial dimensions.

Signed weights are represented in the conductance arrays using differential encoding. Consequently, the 178×32 logical weights of FC1 are expanded into a 178×64 physical conductance matrix, whereas the 32×2 logical weights of FC2 are represented by a 32×4 physical conductance matrix. Owing to the maximum number of addressable rows in an individual array, FC1 is partitioned along its input dimension into three weight submatrices and distributed between PE2 and PE3 (Fig. S8b). PE2 accommodates two 64×64 conductance blocks corresponding to the first 128 FC1 input components, whereas the remaining 50 components are mapped onto a 50×64 block in PE3. The 32×4 differentially encoded FC2 matrix is also mapped onto PE3. Partial outputs from the three FC1 partitions are digitized and accumulated to recover the complete 32-dimensional FC1 output before nonlinear activation.

The input features are first quantized into n -bit unsigned values and decomposed along the bit-width dimension into n binary bit slices (Fig. S8c), such that

$$X_q = \sum_{b=0}^{n-1} 2^b X^{(b)} \quad (4)$$

where $X^{(b)} \in \{0,1\}$ denotes the b -th input bit plane. The bit slices are sequentially applied to the arrays, with each bit plane processed through the corresponding group of tile-read operations. This bit-serial scheme avoids the direct generation of multilevel analogue input voltages and converts the input precision into a controlled number of read cycles. During the b -th read cycle, the positive and

negative-weight arrays generate the column-current vectors I_b^+ and I_b^- , respectively. The two current vectors are digitized separately by the ADCs, and their signed contribution is recovered by digital subtraction:

$$\Delta q_b = Q_{ADC}(I_b^+) - Q_{ADC}(I_b^-) \quad (5)$$

where $Q_{ADC}(\cdot)$ denotes the ADC quantization function. The digitized differential outputs obtained from all bit planes are subsequently left-shifted according to their binary significance and accumulated to reconstruct the signed weighted output:

$$z = \sum_{b=0}^{n-1} 2^b \Delta q_b \quad (6)$$

The trained weights are normalized layer by layer, scaled to the available conductance range and programmed into the corresponding 1T1R cells. Fig. S8d presents the measured conductance distributions of FC1 and FC2 in PE2 and PE3, with the positive and negative weight components arranged in predefined paired columns. The boundaries between array partitions and the weight regions associated with the two network layers remain distinguishable in the conductance maps, confirming that the complete MLP parameters are programmed according to the prescribed differential-encoding and partitioning scheme.

Fig. S8e compares the measured current-domain outputs with the ideal values calculated using the same inputs and target weight matrices. The measured outputs closely follow the ideal response, yielding an RMSE of $8.38 \mu A$ and $R^2=0.96$. The remaining deviations arise from conductance-programming errors, device variations and peripheral readout noise. These measurements verify that the differential mapping, partitioned accumulation and bit-serial execution preserve the matrix-vector computation required for hardware motion prediction.

Supplementary Note 9 CNN-LSTM baseline for single-object prediction

A raw-frame CNN-LSTM model was used as the digital baseline for comparison with the proposed hardware predictor (Fig. S9a). Four consecutive 64×114 grayscale frames were independently encoded by a three-stage CNN with 16, 32 and 64 output channels. Global average pooling produced a 64-dimensional feature vector for each frame, and the resulting four-frame sequence was processed by a single-layer LSTM with 64 hidden units to predict the next-frame target position.

The model was trained for 80 epochs using the mean-squared error of normalized coordinates. Spatial translations, reflections and their combinations were applied identically to the image sequences and coordinate labels for data augmentation, while the original trajectory was reserved for testing. The model with the lowest validation loss was retained (Fig. S9b).

On the original test trajectory, 57.2%, 93.6% and 100% of the predicted positions fall within one, two and three array cells of the reference positions, respectively (Fig. S9c), establishing the

CNN-LSTM as a high-accuracy multiframe, full-frame baseline.

Supplementary Note 10 Hardware implementation of dual-object trajectory prediction

To enable eight-step prediction in dual-object scenes, we constructed a lightweight $178 \times 32 \times 16$ multilayer perceptron (Fig. S10a). The 178-dimensional input comprises 64 row-current and 114 column-current features from the 64×114 memristor array, whereas the 16 outputs represent eight future two-dimensional positions of the intruder UAV. The network was trained for 80 epochs, and both training and validation losses decreased rapidly before approaching stable values (Fig. S10b).

After training, the weights were differentially mapped to the memristor arrays. The 178×32 and 32×16 logical weight matrices were represented by 178×64 and 32×32 physical conductance matrices, respectively. The programming-error distribution is concentrated around zero (Fig. S10c), indicating close correspondence between the programmed and target weights. The mapped network was subsequently used for the eight-step trajectory prediction and proximity-risk assessment in Fig. 5.

Supplementary Note 11 Candidate-conditioned current readout for resolving dual-object ambiguity

During dual-object motion, the two targets can approach one another closely along the horizontal direction, making direct coordinate association ambiguous if only global row and column maxima are used. To resolve this ambiguity, we implemented a candidate-conditioned current-readout scheme. Two horizontal candidates were first identified from the global column-current profile. For each candidate, a fixed local column window centred on the selected peak was then used to acquire a conditioned row-current profile, from which the corresponding vertical coordinate was obtained.

At each frame, the global column-current excess profile was calculated by subtracting the initial-state current from the current readout,

$$I^{col}(x, t) = I^{col}(x, G_t) - I^{col}(x, G_0) \quad (7)$$

where G_t and G_0 denote the current and initial conductance states, respectively. Two local maxima in $I^{col}(x, t)$ were selected as the horizontal target candidates, x_1 and x_2 . Each selected peak defined a candidate-specific 9-column window, Ω_1 and Ω_2 .

For each candidate window, row currents were read out conditionally by summing only the conductance contributions within the corresponding column window. The conditioned row-current excess profile was calculated as

$$I_k^{row}(y, t) = I^{row}(y, G_t, \Omega_k) - I^{row}(y, G_0, \Omega_k), k \in \{1, 2\} \quad (8)$$

where Ω_k denotes the column window centred on the k -th horizontal candidate. The vertical coordinate associated with each candidate was then obtained from the peak of the conditioned row-current profile,

$$y_k = \arg \max_y I_k^{row}(y, t) \quad (9)$$

The resulting paired target coordinates are therefore (x_1, y_1) and (x_2, y_2) , obtained directly from the global column readout and the two candidate-conditioned row readouts without reconstructing the full conductance map.

[Fig. S11a](#) shows four representative frames during the dual-object interaction. The conductance state provides the physical context for the two targets, while the shaded regions indicate the candidate-specific column windows used for conditioned readout. [Fig. S11b](#) shows the corresponding global column-current profiles. In each case, two spatially separated peaks are selected as the horizontal candidates. [Fig. S11c](#) shows the conditioned row-current profiles obtained from the two candidate windows. The peak of each conditioned profile yields the corresponding vertical coordinate. Together, these results demonstrate that candidate-conditioned current readout resolves dual-object coordinate ambiguity and enables direct association of the two-dimensional positions from projection-current measurements.

Supplementary Note 12 CNN-LSTM baseline for dual-object trajectory prediction

A CNN–LSTM model operating on raw image sequences was used as the digital baseline for the dual-object trajectory-prediction task ([Fig. S12a](#)). The network receives four consecutive 64×114 grayscale frames. Each frame is encoded by three 3×3 convolutional layers with 16, 32 and 64 output channels and compressed into a 64-dimensional feature vector. The resulting four-frame feature sequence is processed by a single-layer LSTM with 64 hidden units, and the final hidden state is mapped to 16 outputs corresponding to eight future two-dimensional positions of the intruder UAV.

The model was trained using the mean-squared error of the predicted and reference future coordinates. Across prediction horizons of one to eight frames, the mean coordinate error remains within approximately 2.4–2.8 array cells ([Fig. S12b](#)). In addition, 21.6%, 90.1% and 95% of the predictions fall within one, five and six array cells, respectively ([Fig. S12c](#)). These results establish the CNN-LSTM as an accurate multiframe, full-frame baseline for comparison with the compressed projection-current predictor.

Supplementary Note 13 Memristive mixed-signal hardware platform

The experiments were performed on a modular memristive mixed-signal platform comprising a main control board and plug-in array sub-cards ([Fig. S13](#)). The main board, controlled by an XCKU040 FPGA, provides global timing, data transfer and read/write control, whereas each sub-

card integrates an 8-Kb RRAM chip and an FPGA (XC7K325T) for local addressing and operation sequencing.

Programmable voltages are generated by DACs (AD5532 and DAC5682Z). A hierarchical 8:1, 4:1 and 2:1 multiplexer network reconfigures the array terminals between voltage drive, ground, floating and current-sensing paths. This routing allows the two orthogonal array directions to be alternately configured for driving and sensing, enabling row and column-oriented programming and current readout.

Array currents are acquired through 32 parallel sensing channels and digitized by four eight-channel ADCs (AD7606B), with an AD9650 providing an auxiliary high-speed path. The digitized outputs are returned to the FPGA for conductance extraction, projection-current processing and neural-network computation, enabling device characterization, sensory-state evolution and predictive inference on the same platform.

Supplementary Note 14 Closed-loop resistance targeting by adaptive pulse programming

To achieve reproducible programming of the memristor conductance states, the hardware platform implements a closed-loop resistance-targeting procedure based on write-read-verify operation (Fig. S14). The main board transmits the target resistance and read/write commands to the peripheral circuits and array sub-card, which perform device addressing, pulse application and state readout. After each programming pulse, the device resistance is measured and returned to the main board to determine the subsequent operation.

As shown in Fig. S14a, when the measured resistance is higher than the target value, a SET pulse is applied to decrease the resistance. When the measured resistance is lower than the target, a RESET pulse is applied to increase it. The procedure is repeated until the measured resistance satisfies the prescribed targeting criterion. Because the programming direction is determined by the deviation between the instantaneous and target states, the device can be corrected from either side of the target, reducing the programming error associated with open-loop pulse application.

Fig. S14b,c show representative SET and RESET targeting processes, respectively. During SET programming (Fig. S14b), the positive write voltage is progressively increased and the resistance evolves from a high-resistance state towards a lower target value. During RESET programming (Fig. S14c), the magnitude of the negative write voltage is progressively increased and the resistance evolves from a low-resistance state towards a higher target value.

Supplementary Note 15 Latency and energy comparison for single-object tracking and trajectory prediction

To support the system-level comparison in Fig. 4, the latency and energy consumption of the

graphics-processing-unit (GPU)-based CNN-LSTM baseline and the proposed memristive trajectory-prediction system were evaluated separately. The two systems were assessed using methods consistent with their respective execution mechanisms. For the CNN-LSTM baseline, latency was measured using synchronized CUDA events, whereas GPU-board energy was obtained by integrating the NVML power trace over the inference interval. For the proposed method, the latency and energy were estimated from the hardware execution sequence. In both cases, one complete trajectory prediction was used as the common comparison unit. The system-level comparison in the main text includes only the inference cost during deployment.

The experiment utilized a GPU model of RTX 5070 [2]. Before timing, all inputs were transferred to GPU memory and 20 warm-up runs were performed. The test set containing 236 predictions was then evaluated over 100 repeated runs using mixed precision and a batch size of 32. Latency was measured with synchronized CUDA events. One complete test-set pass required 8.970 *ms*, corresponding to a batch-amortized latency of $T_{GPU} = 38 \mu s$ per prediction. GPU-board power was sampled with NVML at 2 *ms* intervals and integrated over the complete inference interval. One test-set pass containing 236 predictions consumed 0.69 *J*, corresponding to $E_{GPU} = 2.93 mJ$ per prediction.

To evaluate the system-level cost of the proposed memristive visual-prediction system, a complete inference was divided into four sequential stages: FPGA-based temporal differencing and SET/RESET decision generation, sensory-array updating, row-column current readout and memristor array MLP computation. The total latency and energy are expressed as

$$T_{total} = T_{FPGA} + T_{write} + T_{read} + T_{MLP} \quad (10)$$

$$E_{total} = E_{FPGA} + E_{write} + E_{read} + E_{MLP} \quad (11)$$

The individual contributions are calculated below.

For each 64×114 grayscale input frame, with each pixel represented by 8 bits, the FPGA processes 64 pixels in parallel. Each row is divided into 2 blocks, resulting in $64 \times 2 = 128$ processing cycles per frame. At 200 MHz, including one output-pipeline cycle, the latency for temporal differencing and SET/RESET decision generation is

$$T_{FPGA} = (128 + 1)/200MHz = 0.645\mu s \quad (12)$$

The energy per frame is calculated from the post-route dynamic power

$$E_{FPGA} = P_{dyn}T_{FPGA} = 0.4 \times 0.645 = 258 nJ \quad (13)$$

Across evaluated frames, an average of 13 SET and 12 RESET operations were generated per frame. Using a 100-ns SET pulse and a 150-ns RESET pulse, the mean array-update latency under the conservative event-wise programming assumption is

$$T_{write} = 13 \times 100 + 12 \times 150 = 3.1\mu s \quad (14)$$

The device-programming energy was calculated as $E = V^2 GT$. A mean SET voltage of 0.94 V, conductance of 111 μS and pulse width of 100 ns yield 9.81 pJ per SET operation. A mean RESET voltage of 1 V, conductance of 105 μS and pulse width of 150 ns yield 15.75 pJ per RESET operation. The mean device-programming energy is therefore

$$E_{write} = 13 \times 9.81 + 12 \times 15.75 = 0.317 nJ \quad (15)$$

The hardware performs array readout using a 32×32 tile as the basic parallel operation. For column-current projection of the 64×114 memristor array, the 64 input rows and 114 output columns require $N_{col} = 8$ tile operations. For row-current projection, the array is accessed in the orthogonal direction, requiring $N_{row} = 8$ tile operations. The complete row-column projection therefore requires 16 read cycles. With a read duration of 100 ns per tile operation,

$$T_{read} = (8 + 8) \times 100 ns = 1.6 \mu s \quad (16)$$

Partial currents generated from different input tiles are digitized and accumulated digitally. Column-current projection requires 228 ADC conversions, whereas row-current projection requires 256 conversions. Thus, $N_{ADC,front} = 484$. Using 0.97 nJ per ADC conversion^[3]

$$E_{ADC,front} = N_{ADC,front} \times E_{ADC,per} = 469.48 nJ \quad (17)$$

The corresponding input-side DAC activity comprises $N_{DAC,front} = 484$ operations. Using 2.54 nJ per DAC operation^[3]

$$E_{DAC,front} = N_{DAC,front} \times E_{DAC,per} = 1229.36 nJ \quad (18)$$

The total row-column readout energy is therefore

$$E_{read} = E_{ADC,front} + E_{DAC,front} = 1698.84 nJ \quad (19)$$

The $178 \times 32 \times 2$ MLP was implemented using two 64×128 memristor array chips, with each signed weight represented by a differential conductance pair. The 178×32 first layer is mapped onto a 178×64 physical conductance matrix and partitioned along the input dimension into $64 + 64 + 50$ segments. The two 64×64 segments are implemented on PE2, whereas the remaining 50×64 segment and the 32×4 differentially encoded second layer are implemented on PE3.

With a 32×32 tile as the basic array-computing unit, each 64×64 FC1 segment requires 4 tile operations per input bit. The two segments on PE2 therefore require eight operations per bit. The 50×64 FC1 segment on PE3 requires 4 operations per bit, whereas the 32×4 FC2 matrix requires one additional operation. The complete MLP therefore requires 13 tile operations per input bit. With 8-bit bit-serial inputs, $N_{MLP} = 104$ array-read cycles, yielding

$$T_{MLP} = 104 \times 100 ns = 10.4 \mu s \quad (20)$$

The MLP energy consumption includes the memristor-array computation and the peripheral DAC and ADC operations. The crossbar-computation energy was evaluated from the input-

dependent bit-plane activity,

$$E_{array} = \sum_l \sum_{b=0}^7 \sum_{i,j} a_{b,i} V_{read}^2 (G_{l,ij}^+ + G_{l,ij}^-) T_{bit} \quad (21)$$

where $a_{b,i}$ denotes the binary activation of input i in bit slice b , $G_{l,ij}^+$ and $G_{l,ij}^-$ are the differential conductances of layer l , $V_{read} = 0.2V$, and $T_{bit} = 100 \text{ ns}$. For the 8-bit bit-sliced implementation, the mean crossbar-core dynamic energy was $E_{array} = 0.982 \text{ nJ}$ per prediction.

The two 64×64 FC1 segments on PE2 require 2,048 ADC conversions over eight bit planes. The 50×64 FC1 segment on PE3 requires 1,024 conversions, and the 32×4 FC2 matrix requires 32 conversions. Therefore, $N_{ADC,MLP} = 3104$, and

$$E_{ADC,MLP} = N_{ADC,MLP} \times 0.97 \text{ nJ} = 3010.88 \text{ nJ} \quad (22)$$

The corresponding input-side DAC activity is $N_{DAC,MLP} = 3104$ operations, giving

$$E_{DAC,MLP} = N_{DAC,MLP} \times 2.54 \text{ nJ} = 7884.16 \text{ nJ} \quad (23)$$

The total MLP energy is therefore

$$E_{MLP} = E_{array} + E_{ADC,MLP} + E_{DAC,MLP} = 10896.02 \text{ nJ} \quad (24)$$

Combining the four stages gives a total inference latency of

$$T_{total} = T_{FPGA} + T_{write} + T_{read} + T_{MLP} = 15.75 \mu\text{s} \quad (25)$$

The corresponding total energy is

$$E_{total} = E_{FPGA} + E_{write} + E_{read} + E_{MLP} = 12.85 \mu\text{J} \quad (26)$$

Compared with the GPU RTX5070 baseline, the proposed system reduces the inference latency from $38 \mu\text{s}$ to $15.75 \mu\text{s}$, corresponding to a latency reduction of

$$((38 - 15.75)/38) \times 100\% = 58.6\% \quad (27)$$

The inference energy is reduced from 2.93 mJ to $12.85 \mu\text{J}$, corresponding to

$$2.93 \text{ mJ} / 12.85 \mu\text{J} = 228 \quad (28)$$

Therefore, compared to the GPU RTX5070, the proposed system reduces the inference latency by 58.6% and decreases the energy consumption per inference by approximately 228 times.

Supplementary Note 16 Latency and energy comparison for dual-object tracking and trajectory prediction

To evaluate the system-level efficiency of the dual-object prediction task, one complete eight-step trajectory prediction was used as the common comparison unit. The FPGA configuration, 32×32 tile-based array operation, 100 ns read-cycle duration and peripheral-energy parameters follow those described in Supplementary Note 15. The GPU-based CNN-LSTM receives four consecutive 64×114 frames and predicts eight future two-dimensional coordinates. Using the same measurement protocol as in Supplementary Note 15, the GPU baseline requires $T_{GPU} = 52 \mu\text{s}$ and $E_{GPU} = 9.091 \text{ mJ}$ per eight-step prediction.

For the proposed system, FPGA-based temporal differencing is identical to that used in the

single object task, giving $T_{FPGA} = 0.645\mu s$ and $E_{FPGA} = 258nJ$. Across the evaluated dual-object frames, an average of 15 SET and 9 RESET operations were generated per frame. Using 100 ns SET and 150 ns RESET pulses, the mean sensory-array update latency is

$$T_{write} = 15 \times 100ns + 9 \times 150ns = 2.85\mu s \quad (29)$$

The mean SET voltage and conductance were 0.9 V and 104 μS , respectively, whereas the corresponding RESET values were 1.1 V and 122 μS . The device programming energies per operation are therefore $E_{SET} = 8.42 pJ$, $E_{RESET} = 22.14 pJ$. Thus,

$$E_{write} = 15E_{SET} + 9E_{RESET} = 0.33nJ \quad (30)$$

For projection-current acquisition, the global column and row projections require eight tile operations each. Two additional candidate-conditioned row projections are performed using 9 column windows. Each conditioned projection requires two 32×32 tile operations, resulting in $N_{read} = 20$. The corresponding readout latency is $T_{read} = 2\mu s$.

Partial currents generated from different input tiles are digitized separately and accumulated digitally. The global column and row projections require 228 and 256 ADC conversions, respectively, whereas the two candidate-conditioned row projections require 64 conversions each. The complete dual-object readout therefore requires $N_{ADC,front} = 612$. Thus, $E_{ADC,front} = 612 \times 0.97nJ = 593.64nJ$. The corresponding input-side driving requires 520 DAC operations. Using 2.54 nJ per DAC operation, $E_{DAC,front} = 520 \times 2.54nJ = 1320.80nJ$. The total projection-current readout energy is therefore

$$E_{read} = E_{ADC,front} + E_{DAC,front} = 1.914\mu J \quad (31)$$

The eight-step predictor uses a lightweight $178 \times 32 \times 16$ MLP, taking 64 row-current and 114 column-current features as input and producing 16 outputs for eight future two-dimensional coordinates. Differential mapping expands the two layers to 178×64 and 32×32 physical conductance matrices, with the first layer partitioned into 64×64 , 64×64 and 50×64 blocks.

Under the 32×32 tile constraint, the MLP requires 13 tile operations per bit and 104 array-read cycles for 8-bit bit-serial inference, giving $T_{MLP} = 10.4\mu s$. The MLP requires 3,328 ADC conversions and 3,104 DAC operations over the eight input bit planes. The peripheral energies are $E_{ADC,MLP} = 3328 \times 0.97nJ = 3228.16nJ$, $E_{DAC,MLP} = 3104 \times 2.54nJ = 7884.16nJ$. The input-dependent crossbar-core computation contributes $E_{array} = 472pJ$. The total MLP energy is therefore

$$E_{MLP} = E_{array} + E_{ADC,MLP} + E_{DAC,MLP} = 11.113\mu J \quad (32)$$

Combining all four processing stages gives

$$T_{total} = T_{FPGA} + T_{write} + T_{read} + T_{MLP} = 15.90\mu s \quad (33)$$

The corresponding energy consumption is

$$E_{total} = E_{FPGA} + E_{write} + E_{read} + E_{MLP} = 13.29 \mu J \quad (34)$$

Compared with the GPU RTX5070 baseline, the proposed system reduces the inference latency from $52 \mu s$ to $15.90 \mu s$, corresponding to a latency reduction of

$$((52 - 15.90)/52) \times 100\% = 69.4\% \quad (35)$$

The inference energy is reduced from $9091 \mu J$ to $13.29 \mu J$, corresponding to

$$9091 \mu J / 13.29 \mu J = 684 \quad (36)$$

Therefore, compared to the GPU RTX5070, the proposed system reduces the inference latency by 69.4% and decreases the energy consumption per inference by approximately 684 times.

Supplementary Figures

Fig S1. Device electrical characterization.

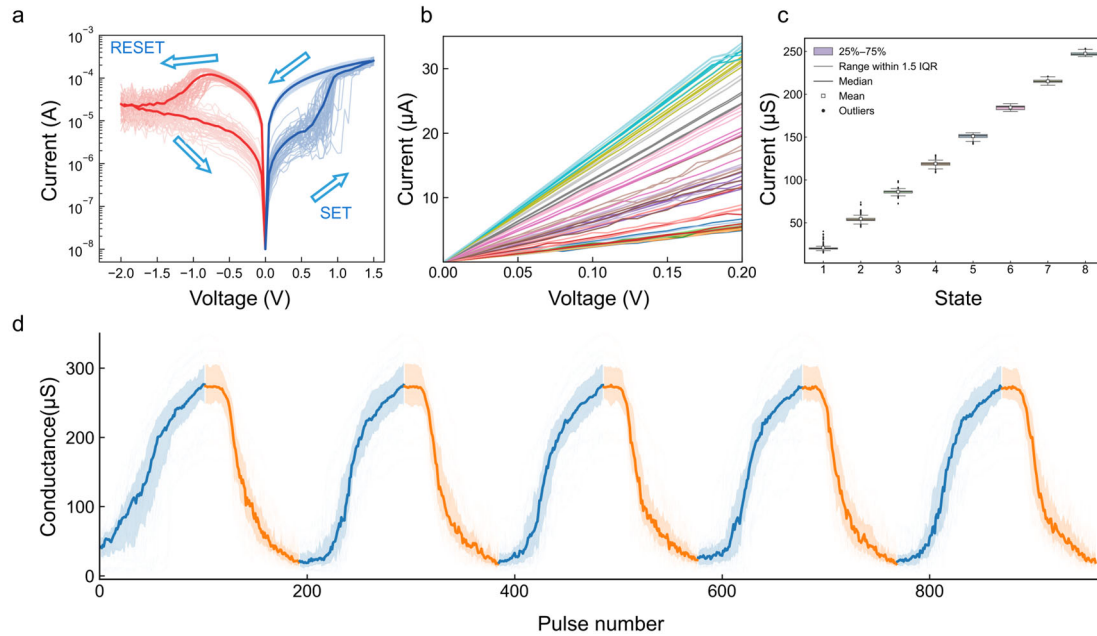


Fig S1. **a** Fifty consecutive bipolar switching cycles under positive SET and negative RESET voltage sweeps. Thin lines show individual cycles and solid lines show the mean. **b** $I-V$ characteristics of 80 progressively programmed conductance states measured from 0 to 0.2 V. **c** Distributions of eight closed-loop-programmed conductance states from 20 to 250 μ S, each measured ten times. Boxes show the interquartile range, centre lines the median, squares the mean and dots the outliers. **d** Potentiation and depression of 32 devices over five SET/RESET cycles. Thin lines represent individual devices, solid lines the median and shaded regions the interquartile range.

Fig S2. Experimental motion-state accumulation in a memristor array.

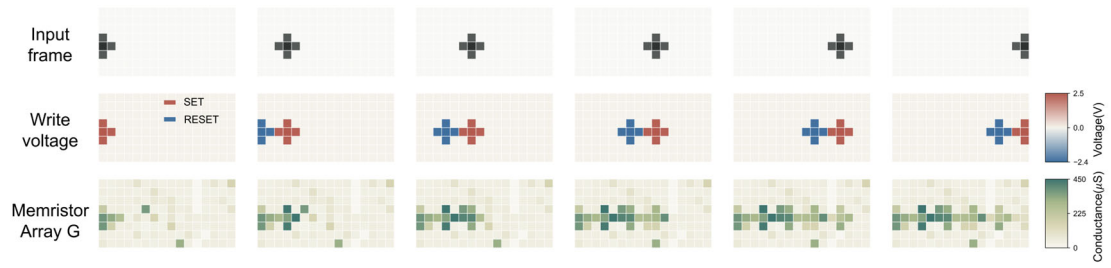


Fig. S2. Six consecutive frames of a target moving from left to right were applied to an 8×16 subarray. The top row shows the input frames, the middle row shows the frame-difference-derived write-voltage maps, where positive and negative voltages produce SET and RESET operations, respectively, and the bottom row shows the measured conductance distributions after each update. The evolving conductance map retains the recent motion history without resetting the array between frames.

Fig S3. Dynamic visual encoding in a memristor array.

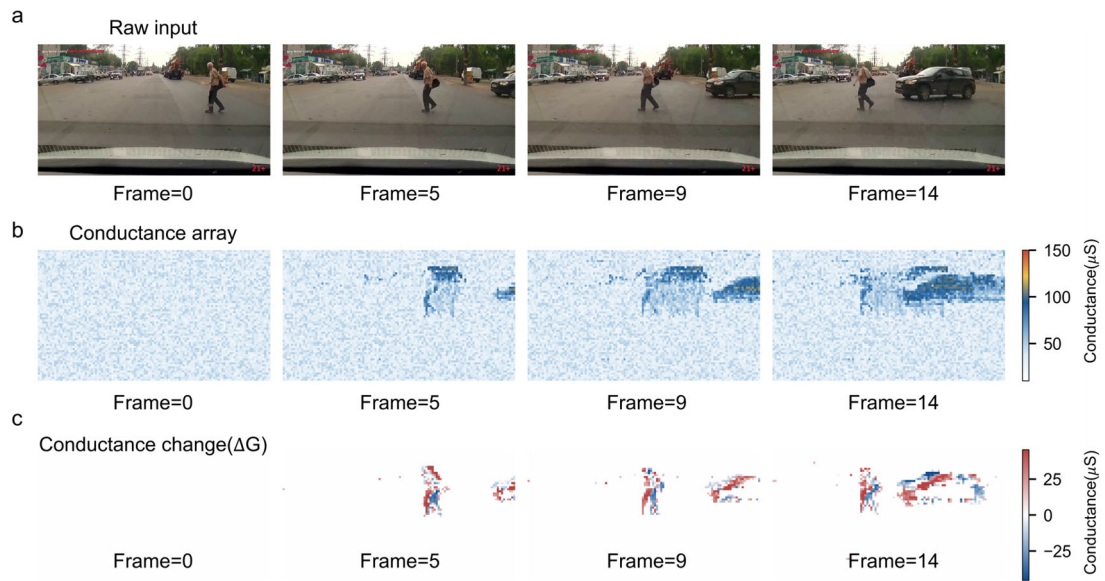


Fig S3. a Representative frames from a dynamic traffic scene containing a moving pedestrian and vehicles. **b** Corresponding conductance maps after in situ array updating, in which the accumulated conductance state preserves recent motion trajectories. **c** Frame-to-frame differential conductance, ΔG , highlighting the instantaneous spatial changes associated with the moving objects.

Fig S4. Comparison between event-based encoding and memristive motion-state accumulation.

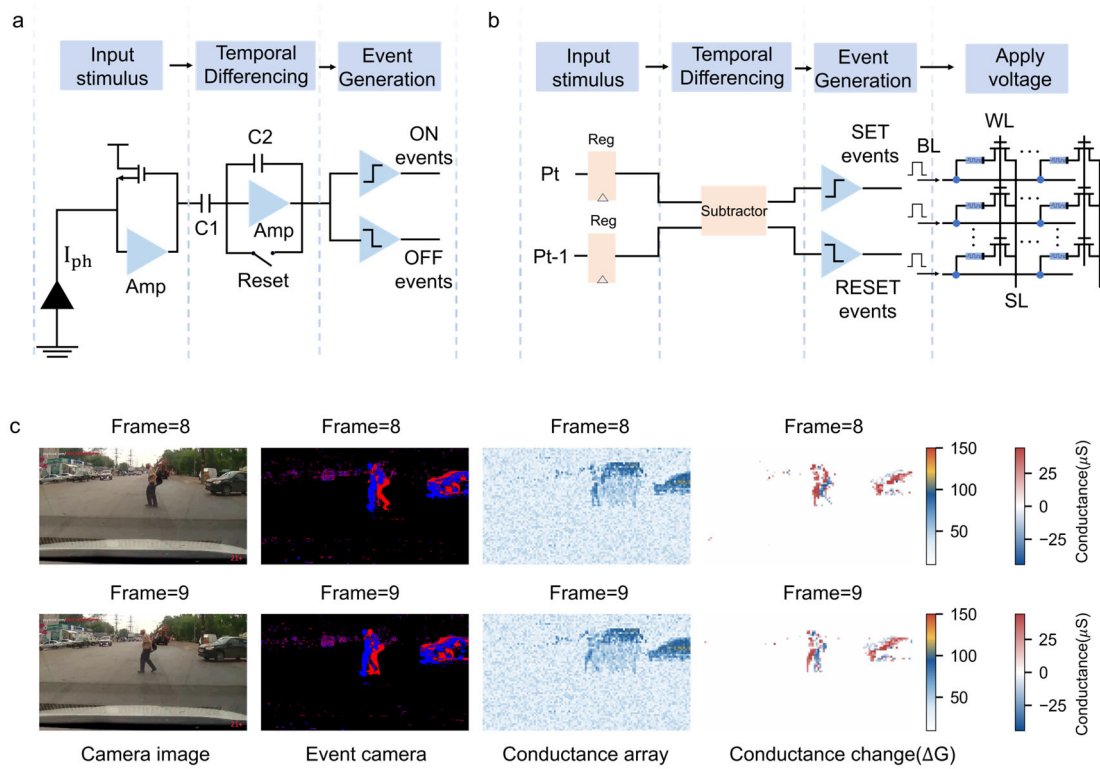


Fig S4. a Event-camera-like processing, in which temporal differencing generates positive and negative events that represent instantaneous changes between consecutive frames. **b** Proposed memristive encoding, in which the same temporal contrasts are converted into SET and RESET voltages and applied to the array, enabling successive motion information to be accumulated in the conductance state. **c** Representative results for two adjacent frames, showing the raw inputs, event maps, accumulated conductance maps and frame-to-frame differential conductance, ΔG . While the event map and ΔG_t capture instantaneous spatial changes, the conductance map G_t additionally retains recent motion history.

Fig S5. Generalization of memristive dynamic visual encoding across diverse driving scenes.

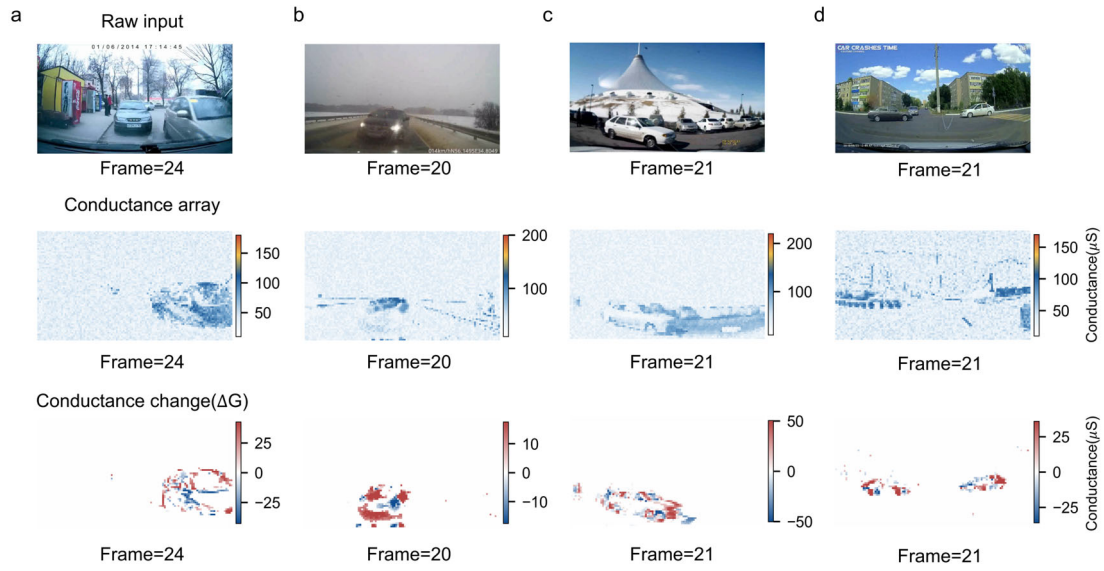


Fig S5. a Snow-covered urban road scene. **b** Low-visibility road scene. **c** Bright open-road scene. **d** Urban traffic scene with multiple moving vehicles. The top row shows representative input frames, the middle row shows the corresponding conductance maps after in situ updating, and the bottom row shows the frame-to-frame differential conductance, ΔG . The accumulated conductance states preserve recent motion trajectories, whereas ΔG highlights the instantaneous spatial changes of the moving vehicles.

Fig S6. Unified hardware implementation for static recognition and dynamic perception.

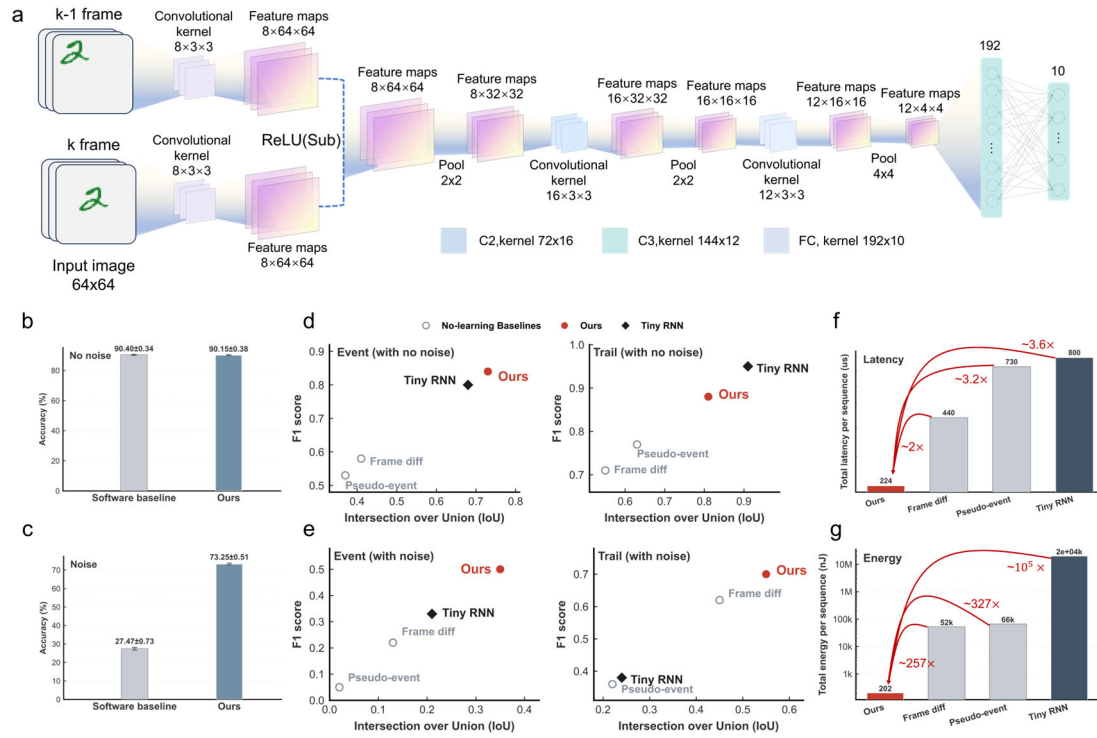
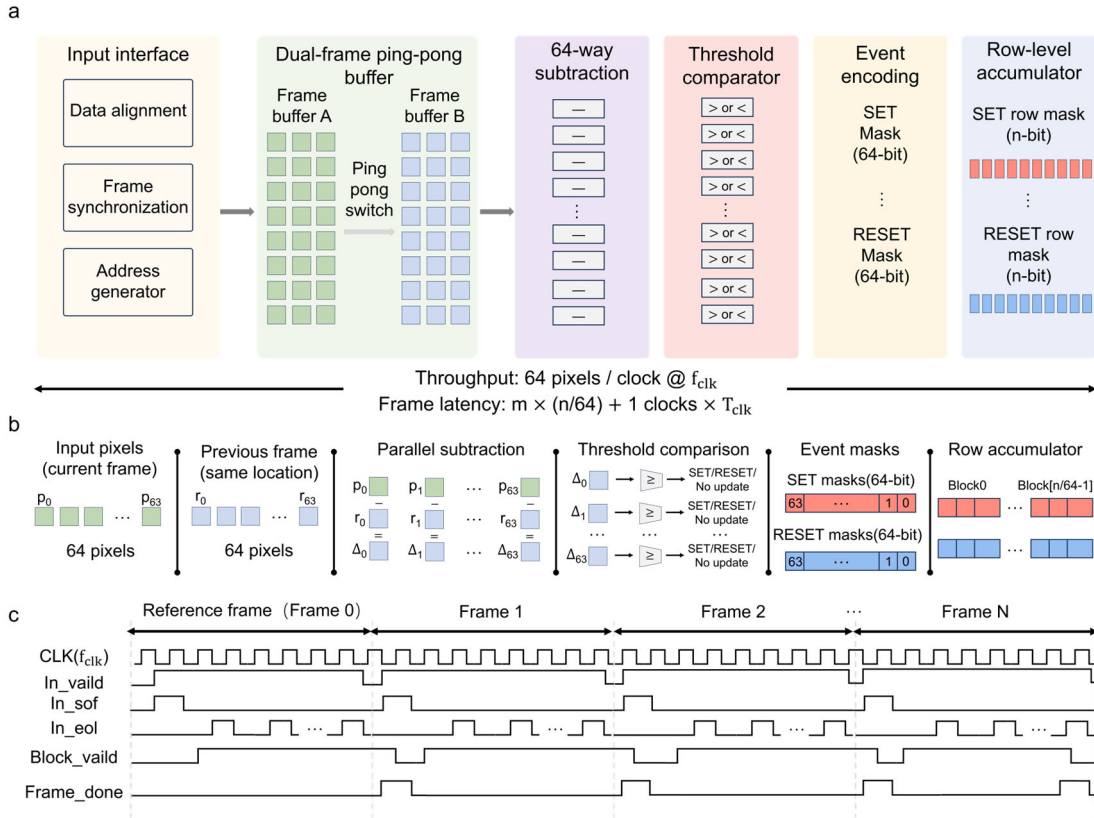


Fig S6. a, Network architecture for processing static and temporal information through a shared sensory-memory hardware pathway. **b,c**, Static-recognition accuracy of the proposed memristive hardware and the corresponding software baseline under noise-free and noisy inputs, respectively. Values are reported as mean $\pm$ s.d. over five repeated evaluations. **d,e**, IoU and F1 score for instantaneous event extraction and accumulated motion-trail reconstruction under noise-free and noisy conditions, comparing the proposed method with frame-difference, pseudo-event and Tiny RNN baselines. **f,g**, Sequence-level latency and energy consumption of the proposed method and the three baselines.

656 **Fig S7. FPGA implementation of parallel temporal-difference encoding.**



657 Fig S7. **a** Architecture comprising the input interface, dual-frame ping-pong buffer, 64-way
658 subtraction, threshold comparison, event encoding and row-level accumulation. The design
659 processes 64 pixels per clock cycle and outputs n-bit SET and RESET masks for each image row. **b**
660 Block-level dataflow for 64 spatially aligned pixels from consecutive frames. Signed temporal
661 differences are classified as SET, RESET or no update and encoded into 64-bit masks, which are
662 concatenated across each row. **c** Timing diagram of the frame-control and output-valid signals. The
663 reference frame initializes the previous-frame buffer without generating events, and frame_done
664 is asserted with the final valid block of each subsequent frame.

Fig S8. Hardware mapping and current-domain validation of the motion-prediction MLP.

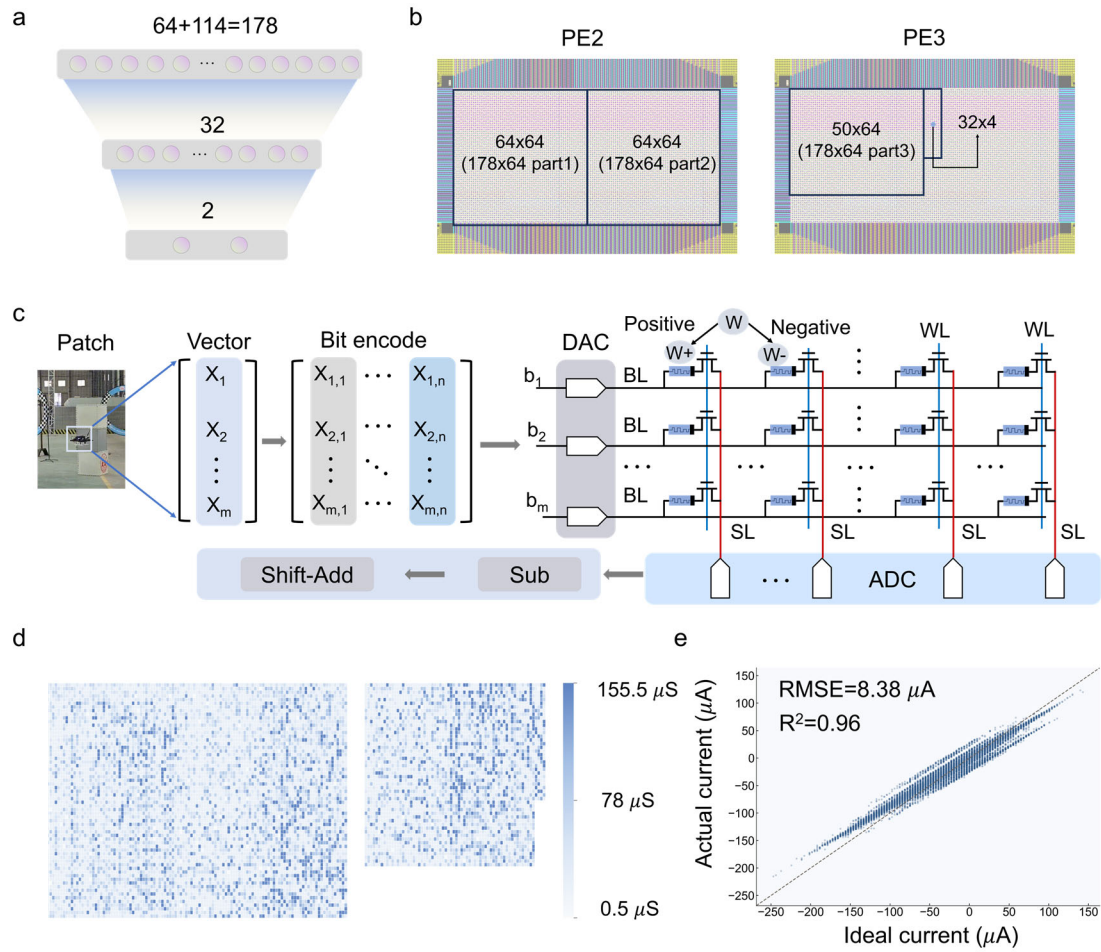


Fig S8. a Architecture of the 178–32–2 MLP used for motion prediction. The 178-dimensional input comprises 64 row-projection features and 114 column-projection features, and the two outputs represent the predicted two-dimensional target displacement. **b** Partitioned mapping of FC1 and FC2 onto PE2 and PE3. **c** Array-computing procedure for the quantized inputs and signed weights. **d** Measured conductance distributions of the FC1 and FC2 weights programmed in PE2 and PE3. **e** Comparison of measured output current and ideal output current of the network.

Fig S9. Architecture and performance of the raw-frame CNN–LSTM prediction baseline.

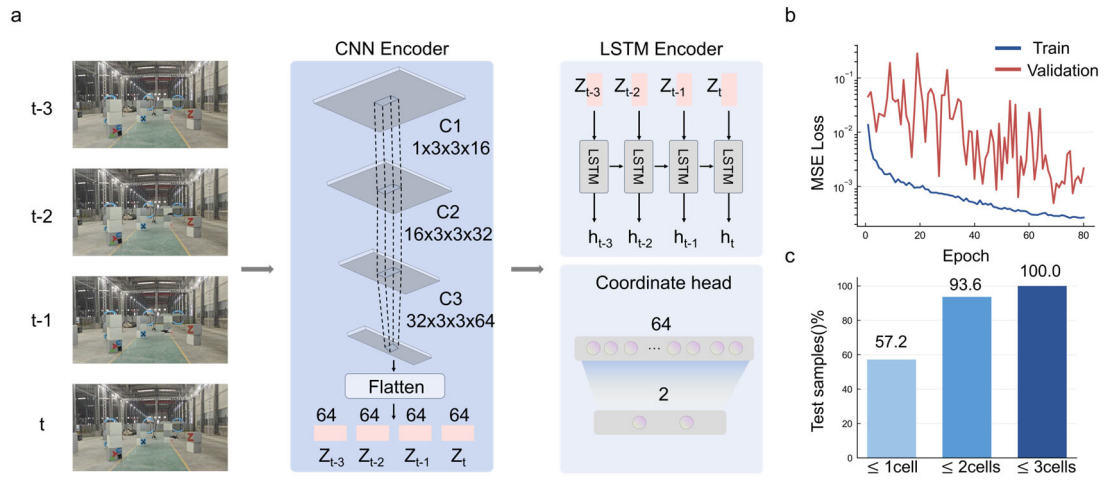


Fig S9. a Four consecutive 64×114 grayscale frames are independently encoded into 64-dimensional spatial features by a three-stage CNN. The four frame features are temporally modelled by a single-layer LSTM, and the final hidden state is mapped by a coordinate head to the target position in the subsequent frame. **b** Training and validation losses over 80 epochs. The vertical axis is shown on a logarithmic scale. **c** Fractions of predictions with errors within one, two and three array cells on the original test trajectory.

Fig S10. Hardware implementation of dual-object trajectory prediction.

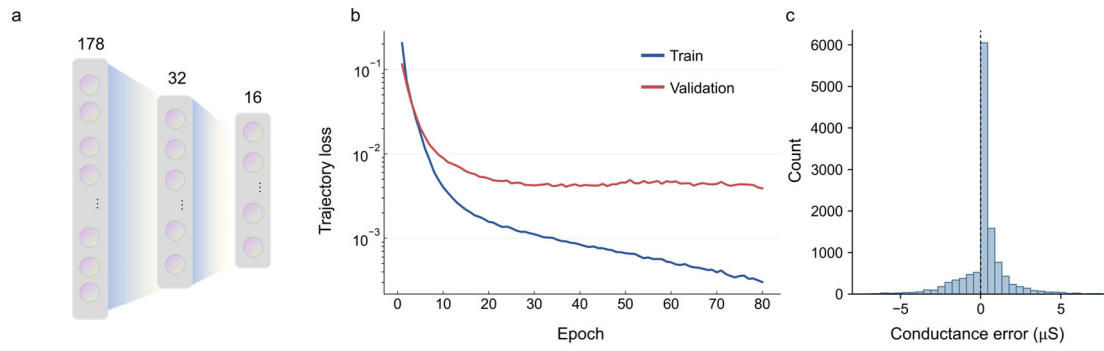


Fig S10. a, Architecture of the multilayer perceptron for eight-step trajectory prediction, using 64 row-current and 114 column-current features as input and producing 16 outputs corresponding to eight future two-dimensional positions. **b,** Training and validation trajectory losses over 80 epochs. **c,** Distribution of conductance-programming errors after differential mapping of the trained weights to the memristor arrays.

Fig S11. Candidate-conditioned current readout for dual-object coordinate association.

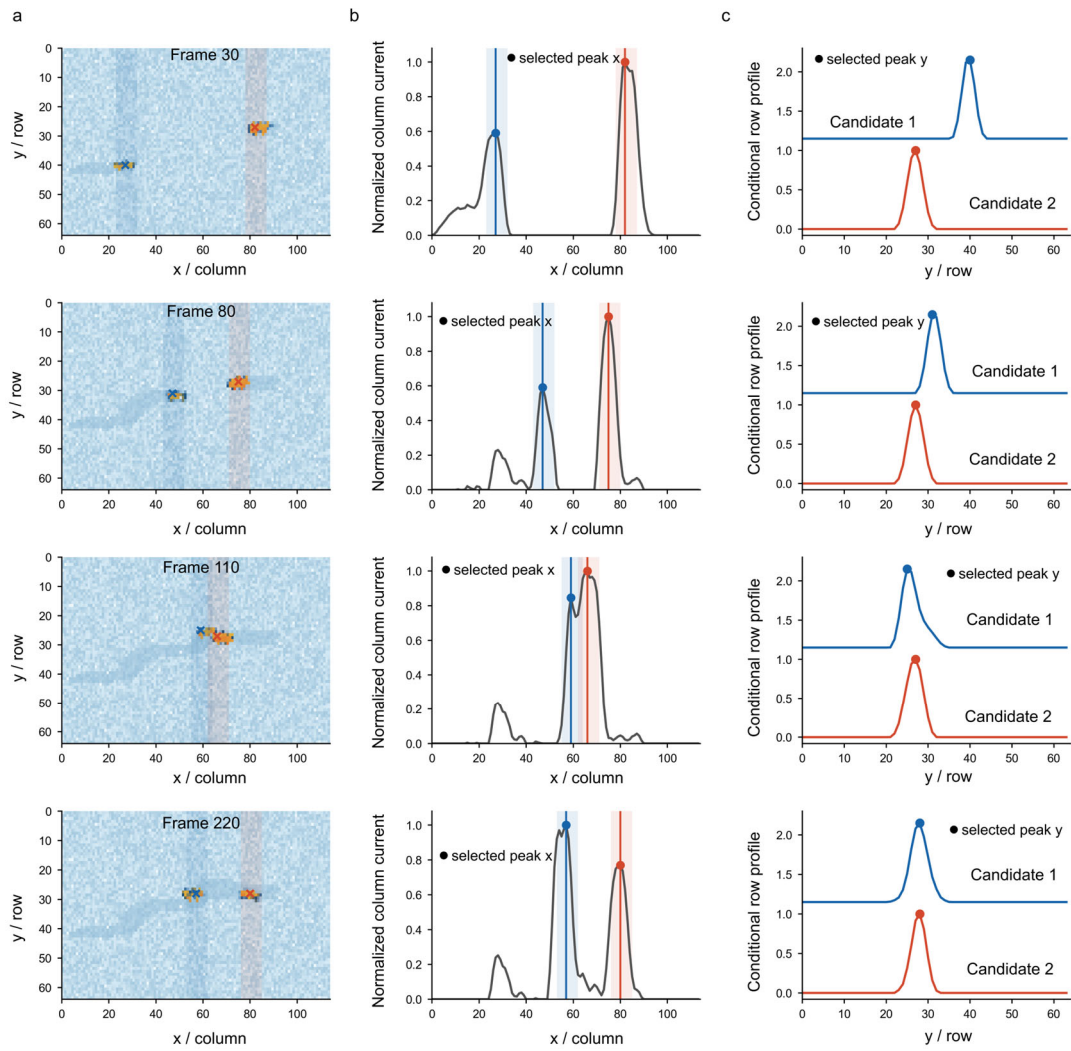


Fig S11. a, Representative conductance states at four frames during dual-object motion, with shaded regions indicating the candidate-specific column windows. **b**, Global column-current profiles used to identify two horizontal candidates; markers indicate the selected column-current peaks. **c**, Candidate-conditioned row-current profiles obtained within the corresponding 9-column windows; markers indicate the selected row-current peaks used to determine the paired vertical coordinates.

Fig S12. CNN–LSTM baseline for multi-step trajectory prediction.

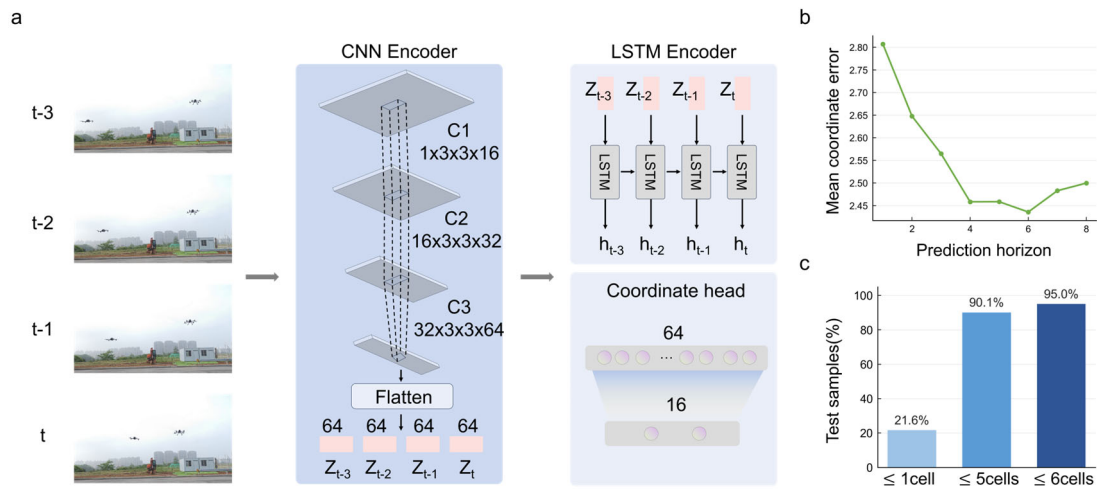


Fig S12. a Network architecture of the raw-frame CNN-LSTM baseline. Four consecutive frames are independently encoded by a three-stage convolutional network and sequentially processed by a single-layer LSTM. The coordinate head produces a 16-dimensional output corresponding to the two-dimensional target coordinates over eight future frames. **b** Mean coordinate error for prediction horizons from one to eight future frames. **c** Fractions of predictions with coordinate errors within 1, 5 and 6 array cells.

Fig S13. Memristive mixed-signal hardware platform.

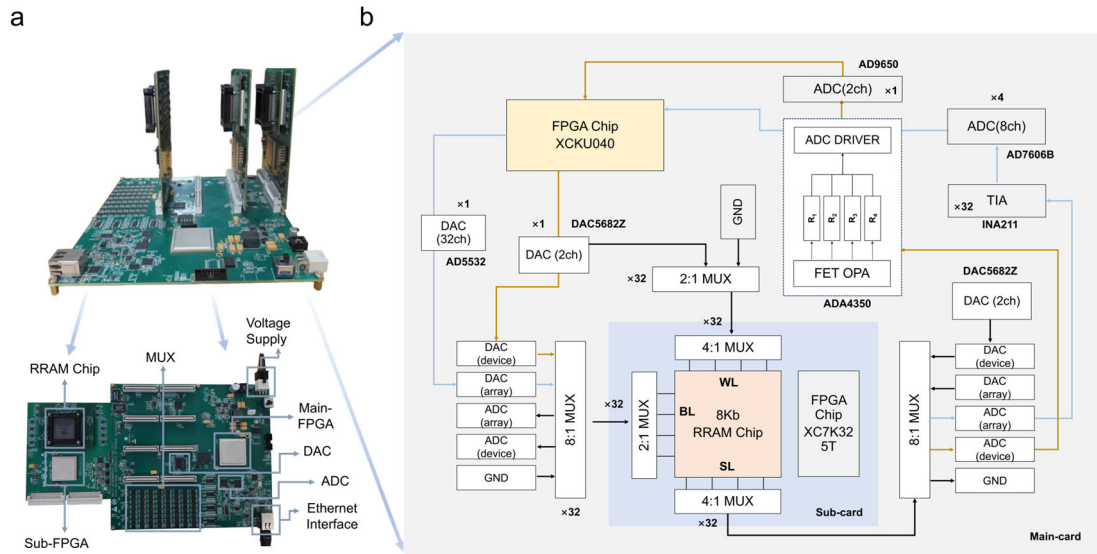


Fig S13. a Photograph of the modular hardware system comprising a main control board and plug-in RRAM array sub-cards. **b** Schematic of the hardware architecture. The XCKU040 FPGA on the main board coordinates the DACs, ADCs, transimpedance amplifiers and switch matrices, whereas the XC7K325T FPGA on each sub-card controls the 8-kbit RRAM chip and local addressing circuitry. The platform supports conductance programming, parallel current readout and current-domain neural-network computation.

Fig S14. Closed-loop target-resistance programming of memristive devices.

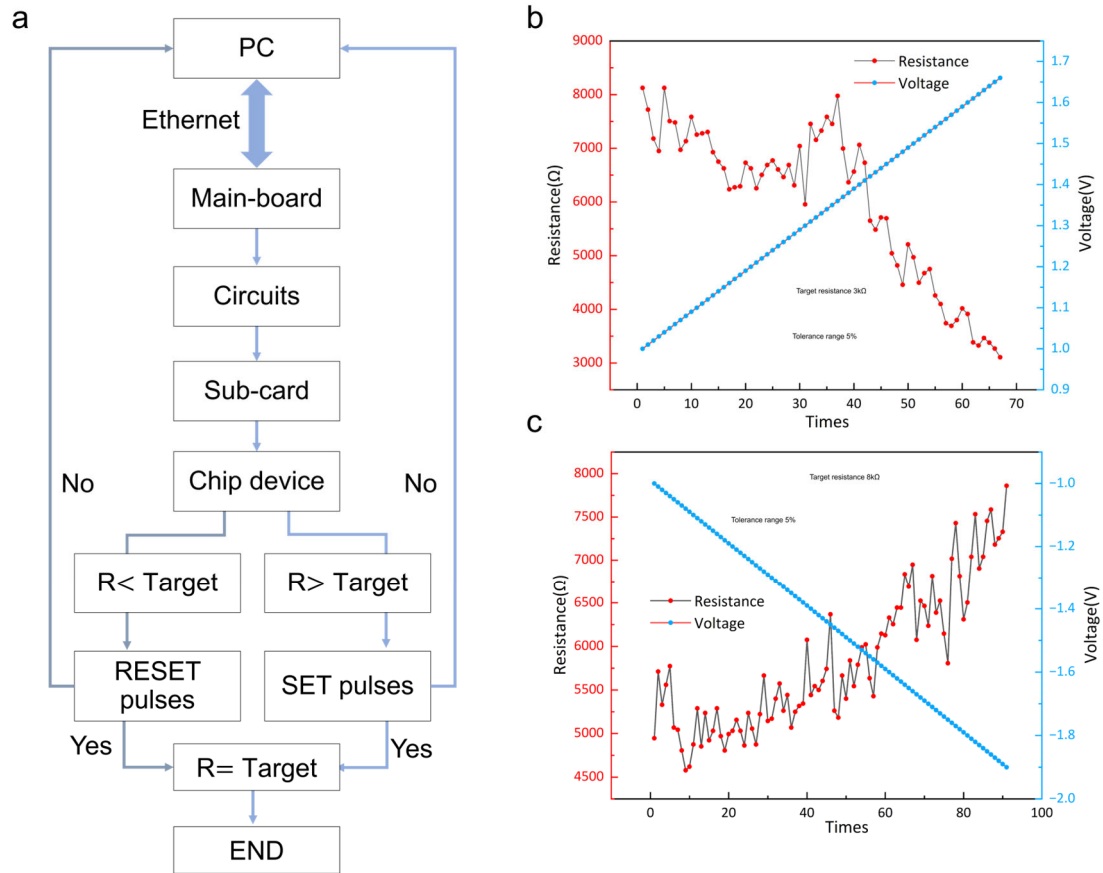


Fig S14. a Write–read–verify procedure for resistance targeting. The measured resistance is compared with the prescribed target after each pulse; SET or RESET pulses are subsequently applied until the target criterion is satisfied. **b** Representative SET targeting process, in which progressively increasing positive write voltages reduce the device resistance towards the target state. **c** Representative RESET targeting process, in which progressively increasing negative-voltage amplitudes increase the device resistance towards the target state.

Supplementary Tables

Supplementary Table 1 System-level latency and energy comparison for single-object trajectory prediction.

System stage	Latency(μs)	Energy(nJ)
FPGA processing	0.645	258
Sensory-array updating	3.1	0.317
Row-column current readout	1.6	1699
Memristor-array MLP	10.4	10896
Proposed system,total	15.75	12853
GPU baseline	38	2.93×10^6
Improvement over GPU	58.6%	228×

Supplementary Table 2 System-level latency and energy comparison for dual-object trajectory prediction.

System stage	Latency(μs)	Energy(nJ)
FPGA processing	0.645	258
Sensory-array updating	2.85	0.326
Row-column current readout	2.00	1914
Memristor-array MLP	10.40	11113
Proposed system,total	15.90	13290
GPU baseline	52	9.091×10^6
Improvement over GPU	69.4%	684 ×

Supplementary Videos

Supplementary Video 1 Hardware future-motion prediction for a single moving object

Hardware demonstration of motion-history accumulation, row-column current projection and future-position prediction on the memristive platform. The evolving conductance state retains recent motion information, while the compressed projection currents are directly processed by the memristive MLP to predict the future target position.

Supplementary Video 2 Hardware multi-object trajectory prediction and proximity-risk assessment

Hardware demonstration of dual-object motion encoding, projection-current-based coordinate association and multi-step trajectory prediction. The predicted intruder trajectory captures the future interaction between the two UAVs and enables image-plane proximity-risk assessment.

References

1. Srivastava, N., Mansimov, E. & Salakhutdinov, R. Unsupervised learning of video representations using LSTMs. In Proceedings of the 32nd International Conference on Machine Learning 843–852 (2015).
2. TechSpot. Nvidia GeForce RTX 5070 Specs. TechSpot (accessed 25 March 2026).
3. Zhang, W. et al. Edge learning using a fully integrated neuro-inspired memristor chip. Science 381, 1205–1211 (2023).