THE ACTION DOESN'T END THE THOUGHT: CONTINUING COMPUTATION IN VISION-LANGUAGE-ACTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Most vision-language-action (VLA) models restart their action expert at every decision, discarding evidence the robot can no longer see and computation the next decision could reuse. We propose *continuing computation*: the expert carries its internal state across decisions and refines it with each new observation. We implement this in LycoReco, a VLA with a Continuous Thought Machine expert, and compare carry and reset experts trained identically except for state lifetime. When earlier cues differ but current inputs are identical, the carried state lets the robot act on information it no longer sees. It also saves expert computation: when replanning at every step on LIBERO, carry with one internal update per observation outperforms reset with twelve. Carry's successive action chunks are more consistent when nothing new is revealed, yet adapt when a remembered target moves. LycoReco is competitive with published VLAs on LIBERO (97.5%) and SIMPLER-WidowX (64.6%) and leads the compared systems on the MIKASA memory tasks (62.6%) without a separate memory module. State lifetime is therefore a consequential architectural choice, and resetting the action expert at every decision should not be the default.

1 INTRODUCTION

Vision-language-action (VLA) models control robots by pairing a pretrained vision-language model (VLM), which encodes the current observation and instruction, with an action expert that turns this encoding into a chunk of future actions. The robot executes part or all of the chunk, receives a new observation, and queries the expert for its next chunk. In almost all current designs, each decision starts the expert from scratch: the computational state that produced the previous chunk is discarded (Black et al., 2025b; Shukor et al., 2025; Kim et al., 2025a). This restart is rarely discussed as a design choice, because stateless feedforward, diffusion, and flow-matching experts already perform well (Kim et al., 2025a; Black et al., 2025a). We ask whether it is the right default: *should an action expert restart its computation at every decision?*

We think not. Restarting discards two things that later decisions can use. The first is **information**. Manipulation is partially observable: the right action often depends on evidence that is no longer in view. Once the cups in MIKASA's shell game (Cherepanov et al., 2026a) cover the ball, for example, no computation on the current image can recover where it is. The second is **computation**. In stages of the task that are relatively static, successive decisions can rely on many of the same intermediate inferences, such as localizing the target and planning the path of an ongoing reach. The internal state that produced one decision may therefore already contain much of the computation the next one needs. Model-predictive control exploits this overlap by warm-starting each optimization from the previous solution (Diehl et al., 2005), so that even one iteration per step can track a good solution. Losing information limits what a policy can know, while discarding computation makes every decision redo work the last one already did.

Existing VLAs keep the past in a memory module (Shi et al., 2026; Torne et al., 2026; Cherepanov et al., 2026b), iterate within each decision (Tur et al., 2026; Shen et al., 2026), or carry actions or their history forward (Høeg et al., 2025; Black et al., 2025c; Zhao et al., 2023; Hu et al., 2026), but no VLA action expert iterates a fixed-size internal state across observations, and none isolates that

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

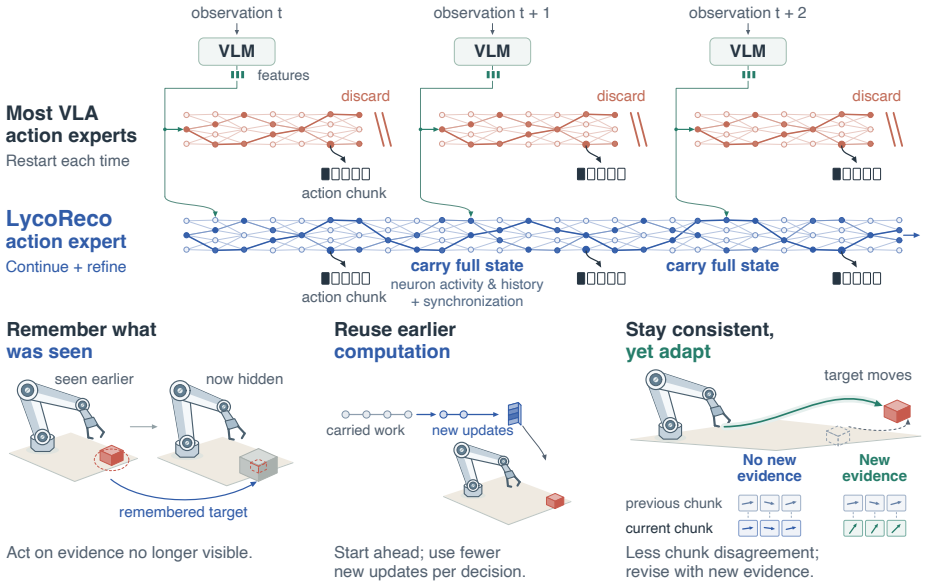


Figure 1: **Action generation as continuing computation.** Most VLA action experts restart their computation at every decision and discard it after emitting an action chunk. LycoReco's expert carries its full state to the next observation and keeps refining it. The bottom row shows three predictions of this view, tested in Sections 4.3–4.5.

choice with matched controls. To recover both losses with one mechanism, we propose *action generation as continuing computation*: the expert starts each observation from the state that produced its previous chunk, updates it for a few internal steps (*ticks*), and reads out the next chunk (Figure 1). Carrying a state is not new in itself (Hausknecht & Stone, 2015; Graves, 2016; Mandlekar et al., 2022; Darlow et al., 2025). What we change is the default for VLA action experts, many of which already iterate on each observation: they should continue that iteration instead of restarting it.

Continuing computation makes three testable predictions (Figure 1). The carried state should (i) *retain* evidence that is no longer visible, and (ii) *reuse* earlier computation, so that the expert acts as well as a restarted one with fewer ticks per observation, even on tasks that need little memory. Since each chunk builds on the computation behind the last, successive chunks should (iii) *remain consistent* when nothing changes, yet still adapt when the evidence does change (Wen et al., 2020). We test each prediction by changing a single factor while keeping the model, data, and training budget fixed: the state's *lifetime*. A *carry* expert keeps its state for the next observation, and a *reset* expert restarts from a learned initial state at every decision.

We realize this design in LycoReco, a VLA whose action expert is a 3M-parameter Continuous Thought Machine (CTM) (Darlow et al., 2025), attached to PaliGemma (Beyer et al., 2024). CTM suits this design because its state is a record of its own recent activity (Section 3.2). The same interface applies to a standard recurrent Transformer (RT); Appendix E compares carry and reset RT controls on MIKASA.

Our contributions are as follows:

- **Continuing computation for action experts.** We propose that a VLA action expert should continue the computation behind its previous action chunk instead of restarting, and we make the lifetime of its computational state an explicit design choice, separate from computation per observation, chunk horizon, and replanning cadence.
- **Evidence for continuing.** Comparing separately trained experts that differ only in state lifetime, we find that carrying the state (i) retains information the robot no longer sees, (ii) needs far less expert computation per observation than restarting, and (iii) keeps successive chunks more consistent, yet still changes them when the evidence changes.

- **LycoReco, a compact CTM VLA.** With a 3M-parameter action expert, it is competitive with published VLAs, reaching 97.5% on LIBERO and 64.6% on SIMPLER-WidowX, and leads the compared systems on average over five MIKASA memory tasks without a separate memory module (62.6% vs. 41.2%). With fewer parameters, a carried CTM expert also outperforms a carried recurrent Transformer expert on MIKASA.

2 Related Work

Memory in VLAs. Memory-augmented VLAs retrieve context from a consolidated memory bank (Shi et al., 2026), keep multi-scale memory (Torne et al., 2026), carry recurrent memory tokens through the backbone (Cherepanov et al., 2026b). RB-VLA maintains a recursive belief and warm-starts action sequences (Bagaria et al., 2026); EvoScene jointly denoises actions and scene outputs, carrying the latter into the next VLM call (Zhang et al., 2026). LycoReco carries the internal recurrent state updated by each expert tick. This can be complementary: memory at the VLM level can hold episodic or semantic context over long horizons, while the carried state keeps the expert's own computation between decisions.

Iterative action experts. Diffusion and flow heads refine each chunk over several denoising steps (Chi et al., 2023; Black et al., 2025b; Shukor et al., 2025), and recurrent-depth VLAs refine a latent state over several iterations (Tur et al., 2026; Shen et al., 2026). Both restart at every decision and discard the refined state once the chunk is read out. Recurrent robot policies keep their state but update it once per observation: BC-RNN (Mandlekar et al., 2022) and RoboFlamingo's LSTM head over per-step VLM features (Li et al., 2024) are the single-tick case of our formulation. Existing VLA experts thus tie the lifetime of their state to how they compute: iterative experts iterate but restart, and recurrent policies carry their state but do not iterate. Outside robotics, adaptive-computation RNNs and CTM's reinforcement-learning agents already do both (Graves, 2016; Darlow et al., 2025). We make the lifetime a design choice of its own, so an expert can iterate on each observation and continue from where the previous one ended.

Consistency across chunks. Several chunked policies encourage agreement between successive chunks. Temporal ensembling and bidirectional decoding reconcile overlapping chunks (Zhao et al., 2023; Liu et al., 2025), real-time chunking conditions each chunk on the actions already committed (Black et al., 2025c), and Streaming Diffusion Policy keeps partially denoised future actions across steps (Høeg et al., 2025). Diffusion Forcing combines per-token denoising with a recurrent latent that summarizes observed history (Chen et al., 2024). AR-VLA likewise avoids resetting at every observation, carrying a rolling cache of recent action and proprioception tokens (Hu et al., 2026). We carry the VLA expert's internal tick state, and more consistent chunks follow as a consequence rather than as a design goal.

3 Method

3.1 Continuing the expert's state

At each control step t, a VLA receives an observation o_t: camera images, the instruction, and the robot's proprioception. Its action expert keeps an internal state h and updates it several times on each observation. We call one update a *tick*. Each tick applies a transition F_θ that attends to the VLM's features of o_t. After T ticks, a readout R_θ maps the state to a chunk $\hat{A}_t$ of H future actions. Writing h_t^k for the state after k ticks on observation o_t, the expert computes

$$h_t^0 = h_{t-1}^T, \qquad h_t^{k+1} = F_\theta(h_t^k, o_t), \qquad \hat{A}_t = R_\theta(h_t^T). \tag{1}$$

The first equation carries the state: the ticks on o_t start where the ticks on o_{t-1} ended, and only the first observation of an episode starts from a learned initial state h_{init}.

The robot executes the first K actions of each chunk before it adopts the next; we call each adopted readout a *decision*. Carrying the state changes where each observation's ticks start, not how many ticks run (T), how far a chunk looks ahead (H), or how many of its actions execute (K).

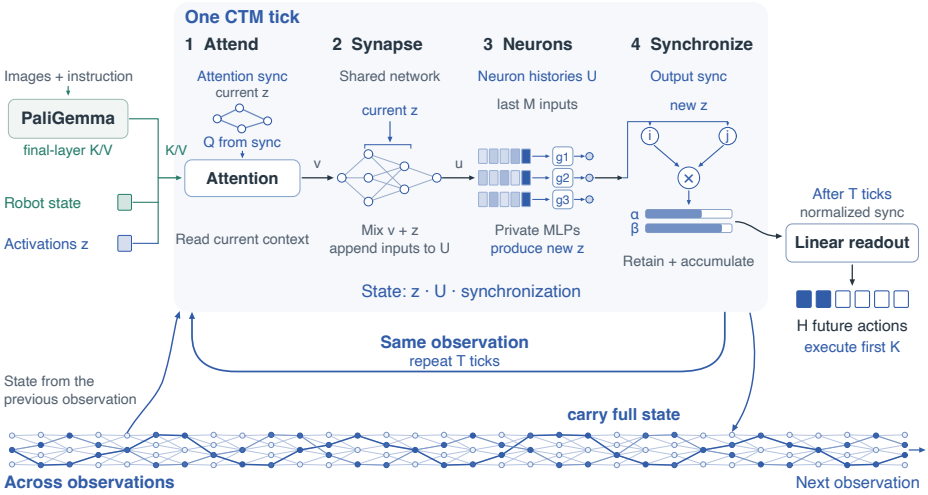


Figure 2: **The architecture of LycoReco.** For each observation, the expert repeats one tick T times. (1) Attend: a query from the current synchronization reads PaliGemma's keys and values and the robot state into v. (2) Synapse: a shared network maps v and the post-activations z to new pre-activations u, appended to the history U of the last M ticks. (3) Neurons: private networks $g_1, g_2, \ldots$ map each neuron's history to its new value in z. (4) Synchronize: the new z updates the accumulators α, β of neuron pairs (i, j) (Eq. 2). The state $h = (z, U, \alpha, \beta)$ then carries to the next observation. A linear readout of the synchronization predicts H actions, and the robot executes the first K.

Equation (1) contains two familiar designs. With $T = 1$, it is an ordinary recurrent policy that updates once per observation (Hausknecht & Stone, 2015; Mandlekar et al., 2022). With $h_t^0 = h_{\text{init}}$ at every decision, it is a *reset* expert, which, like recurrent-depth VLAs (Tur et al., 2026; Shen et al., 2026), iterates at every decision but starts afresh each time.

3.2 LycoReco: a VLA with a compact CTM expert

LycoReco attaches an action expert to PaliGemma-3B (Beyer et al., 2024), and its transition F_θ is a Continuous Thought Machine (CTM) (Darlow et al., 2025) with D neurons. We use a CTM because its whole state is a record of its recent computation. Each neuron keeps a short history of its recent inputs, and pairs of neurons accumulate how they co-activate over ticks, with a retention learned for each pair. Carried to the next observation, this record lets the expert build on the computation behind its previous chunk (Figure 2). The VLM encodes only the current observation, so anything the policy remembers must live in the carried state. The action expert processes every control observation, including those between decisions when $K > 1$.

Formally, the state $h = (z, U, \alpha, \beta)$ contains the neurons' post-activations $z \in \mathbb{R}^D$, each neuron's pre-activations from the last M ticks, $U \in \mathbb{R}^{D \times M}$, and synchronization accumulators α, β for two fixed sets of randomly chosen neuron pairs, an *attention set* and an *output set*. For a pair (i, j) with learned retention $r_{ij} \in (0, 1]$, the accumulators update as

$$\alpha_{ij} \leftarrow r_{ij}\alpha_{ij} + z_i z_j, \qquad \beta_{ij} \leftarrow r_{ij}\beta_{ij} + 1, \qquad S_{ij} = \alpha_{ij}/\sqrt{\beta_{ij}}, \tag{2}$$

where S_{ij} is the synchronization feature. Each tick has four steps.

1. *Attend.* The attention set applies Eq. (2) with the current z, and its synchronization features form a query over PaliGemma's final-layer keys and values for o_t, a proprioception token, and a token formed from z.
2. *Synapse.* A shared network maps the attention output and z to new pre-activations, which are appended to U.
3. *Neurons.* Each neuron's private network maps its last M pre-activations to a new post-activation z_i.

4. *Synchronize.* The output set applies Eq. (2) with the new z.

The readout R_θ is a linear map from the output set's synchronization features to the H actions of the chunk. Because these features accumulate over ticks, including ticks on earlier observations, the chunk depends on the neurons' recent activity, not only on the current observation.

3.3 Training

The expert regresses continuous action chunks with a masked L1 loss in normalized action coordinates. We train carry experts on chronological windows of several observations: each window continues the state left by the previous window of the same episode, with gradients truncated at window boundaries. Appendix B gives window lengths and benchmark-specific configurations.

4 Experiments

We first introduce the benchmarks and show that LycoReco is competitive with published VLAs on them. We then further ask a series of questions that evaluate the mechanics of continuing computation.

1. Does the carried state retain task-relevant information that is no longer available in the current observation? (Section 4.3)
2. Does continuing computation reduce the compute needed per decision, even when the task does not require memory? (Section 4.4)
3. Can carrying state make consecutive action chunks more consistent while preserving responsiveness to new evidence? (Section 4.5)
4. Does continuing computation remain beneficial when the action expert runs more frequently than the VLM? (Section 4.6)

4.1 Setup

Benchmarks. We evaluate on three benchmarks covering general manipulation, real-to-sim transfer, and memory-dependent control. For each benchmark, H denotes the predicted action-chunk length and K the number of actions executed from each chunk. We follow each benchmark's own success criterion.

- **LIBERO** (Liu et al., 2023) evaluates general manipulation across the Spatial, Object, Goal, and Long suites. We use a single policy for all 40 tasks, with $H = 20$ and $K = 10$. Success is determined by each task's standard criterion.
- **SIMPLER-WidowX** (Li et al., 2025) evaluates policies trained exclusively on real world WidowX (robot) demonstrations from BridgeData V2 (Walke et al., 2023). Evaluation covers four tasks in simulated scenes matched to real-world setups, with simulated success serving as a proxy for real-robot success. We use $H = 16$ and $K = 4$, and measure success at the final step of each episode.
- **MIKASA** (Cherepanov et al., 2026a;b) evaluates memory-dependent control on the five tasks used by MemoryVLA (Shi et al., 2026): ShellGameTouch (SGT), InterceptMedium (IM), and RememberColor with three, five, or nine colors (RC3/5/9). In SGT and RC3/5/9, the robot must reach a target specified by a cue that is hidden before it acts. In IM, it must infer a rolling ball's velocity from its past positions to direct the ball toward a target. We use $H = 4$ and $K = 1$, and count an episode as successful if the task's success criterion is met at any step.

Settings. Section 4.2 compares LycoReco with the published results of other VLAs. LycoReco adapts PaliGemma (Beyer et al., 2024) to each benchmark, on LIBERO and SIMPLER-WidowX in two stages: PaliGemma is first fine-tuned with a reset expert, then frozen while the carry expert trains (Appendix B). The compared systems differ in perception, data and training, so performance differences are not solely attributable to architectural differences. Sections 4.3–4.6 isolate state lifetime. For comparison experiments, reset and carry experts are trained identically (Appendix D) and differ only in whether the state survives to the next observation. For experiments trained with frozen PaliGemma, because the base policy is never trained on robot data, absolute success is lower, so they are not directly comparable.

Table 1: **With a 3M-parameter expert, LycoReco is competitive on LIBERO and SIMPLER-WidowX and leads on the MIKASA memory tasks.** Native success (%). Sources and parameter counts are in Appendix C. Bold: best success in each column; underline: second best.

LIBERO					
Method	Spatial	Object	Goal	Long	Mean
RD-VLA (Tur et al., 2026)	92.0	99.0	96.0	84.8	93.0
MemoryVLA (Shi et al., 2026)	98.4	98.4	96.4	93.4	96.7
HAMLET (Koo et al., 2026)	**99.0**	**100.0**	**99.2**	92.2	**97.6**
ContextVLA (Jang et al., 2025)	98.3	99.2	95.6	90.2	95.8
OpenVLA-OFT (Kim et al., 2025a)	97.6	98.4	97.9	<u>94.5</u>	97.1
$\pi_{0.5}$ (Black et al., 2025a)	<u>98.8</u>	98.2	<u>98.0</u>	92.4	96.9
LycoReco	98.0	<u>99.4</u>	97.8	**94.6**	<u>97.5</u>
BridgeData V2 → SIMPLER-WidowX					
Method	Spoon	Carrot	Stack	Eggplant	Mean
MemoryVLA (Shi et al., 2026)	**75.0**	<u>75.0</u>	37.5	**100.0**	**71.9**
HAMLET (Koo et al., 2026)	**75.0**	62.5	16.7	**100.0**	63.5
ContextVLA (Jang et al., 2025)	60.7	**81.3**	**78.7**	62.0	<u>70.7</u>
AR-VLA (Hu et al., 2026)	**75.0**	54.2	20.8	<u>95.8</u>	61.5
OpenVLA-OFT (Kim et al., 2025a)	12.5	4.2	4.2	**100.0**	30.2
$\pi_{0.5}$ (Black et al., 2025a)	49.3	64.7	44.7	69.7	57.1
LycoReco	<u>62.5</u>	58.3	<u>58.3</u>	79.2	64.6
MIKASA: five memory tasks					
Method	SGT	IM	RC3	RC5	RC9	Mean
MemoryVLA (Shi et al., 2026)	**88.0**	24.0	44.0	<u>30.0</u>	<u>20.0</u>	<u>41.2</u>
Octo (Octo Model Team et al., 2024)	46.0	<u>39.0</u>	45.0	17.0	11.0	31.6
OpenVLA-OFT (Kim et al., 2025a)	47.0	14.0	<u>59.0</u>	16.0	6.0	28.4
π_0 (Black et al., 2025b)	33.0	**42.0**	35.0	22.0	15.0	29.4
LycoReco	<u>80.0</u>	35.0	**93.0**	**73.0**	**32.0**	**62.6**

Measurements. All evaluations run in synchronous simulation: the simulator waits for the policy, so we measure computation rather than reaction time (Appendix C.2). We train one seed per expert unless noted. All intervals are 95% intervals over evaluation scenes, so they do not include training variance; paired conditions, such as reset and carry on the same scene, are resampled together.

4.2 Is LycoReco competitive with published VLAs?

With a 3M-parameter expert and no separate memory module, LycoReco is competitive with published VLAs and leads them on memory tasks. It reaches 97.5% on LIBERO and 64.6% on SIMPLER-WidowX. On the five MIKASA tasks it leads with 62.6% against at most 41.2% from MemoryVLA, which adds a dedicated memory module whereas LycoReco has no such mechanism. Trained by the same recipe but restarted at every decision, LycoReco averages only 25.8% (Appendix D.6). Because these tasks require acting on cues that are no longer visible, we attribute the gap between carry and reset to the carried internal state. The next experiment makes this more certain.

4.3 Does the carried state retain task-relevant information?

The carried state retains task-necessary information from the past. We test this on RC3: the robot sees a cube of one color, which is then hidden, and must later touch the cube of that color among red, lime, and blue. The robot is held still until the cubes appear, so we can replay each of 50 scenes three times with only the cue changed; when the cubes appear, the policy's inputs are identical (Figure 3). This test needs a simulator: a physical scene cannot be replayed with identical inputs. The VLM sees only the current observation, so only the carried state can tell the replays apart. A deterministic memoryless policy therefore acts identically in all three, so the first cube it

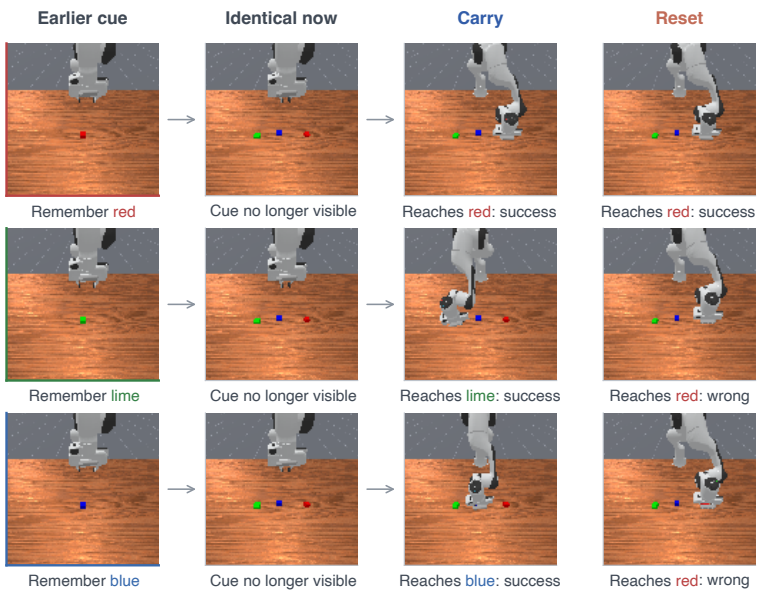


Figure 3: One RC3 scene replayed with three cues, one per row; the observation when the cubes appear is identical in all three. Carry reaches the cued cube in every row. Reset cannot tell the rows apart: it reaches the red cube each time in response to this scene's initial conditions, and is right only when the cue was red.

Table 2: **Carry reaches the cued cube; reset is at chance (RC3 paired cues).** Each of 50 scenes is replayed once per cue. First cube reached: within 5 cm of the gripper. Succeeded: reaches the cued cube and stays. Upper rows: the MIKASA model of Table 1 and its matched reset expert. Lower rows: experts on frozen PaliGemma features; + full history: a reset expert whose CTM also reads the VLM features of every earlier top-camera frame of the episode; Carry RT: a recurrent Transformer expert that also carries its state (Appendix E).

	First cube reached (of 150)			Succeeded
Expert	Correct	Wrong	None	(of 150)
Fine-tuned PaliGemma				
Carry (LycoReco)	135	10	5	133
Reset	50	100	0	50
Frozen PaliGemma				
Carry	121	5	24	119
Reset	39	78	33	37
Reset + full history	80	48	22	78
Carry RT	94	36	20	94

reaches is correct in at most one: at most 50 of 150 replays. This bound applies to the first cube reached, not eventual completion.

For reset, the first cube reached is invariant across cues and correct in exactly one replay of each scene. Carry reaches the cued cube first in 135 of 150 replays and completes 133 (Table 2). In 38 of the 50 scenes, carry both reaches the cued cube first and completes in all three replays, which no deterministic memoryless policy can.

Retaining a cue is expected of any recurrent state. So we test carry vs other implementations of memory. With frozen PaliGemma features, carry averages 47.6% over the five MIKASA tasks, a reset expert that reads the VLM features of every earlier frame 38.4%, a carry Recurrent Transformer (RT) 34.8%, and reset 20.4% (Appendix E). Carry also leads on the paired cues (Table 2, lower rows). The next test asks whether carrying also reuses computation when the current observation contains everything the task needs.

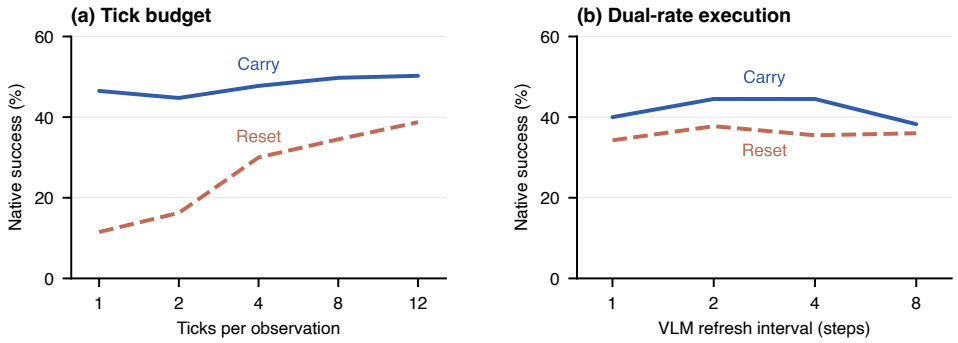


Figure 4: Carry needs far fewer ticks and has stable performance with less frequent PaliGemma updates (LIBERO, 400 episodes per point). (a) Success versus ticks per observation; each point is a separately trained expert. (b) A dual-rate trained carry–reset pair: PaliGemma refreshes every R steps, while the expert runs $T = 12$ ticks at every step (Section 4.6).

4.4 Does continuing save compute?

Carry needs far fewer ticks: with one tick per observation, it outperforms reset with twelve. On LIBERO the current observation largely determines the correct action, so a carried state has little to remember. What continuing can still change is what the expert's ticks accomplish: a reset expert spends each observation's ticks rebuilding a chunk from its initial state, whereas a carry expert contains some prior about the chunk through its internal state. If this matters, carry should act as well as reset with fewer ticks. For every budget $T \in \{1, 2, 4, 8, 12\}$ we train separate reset and carry experts at that fixed budget (Appendix D.5).

Carry changes little with the budget, from 46.5% at one tick to 50.3% at twelve, whereas reset rises from 11.5% to 38.8% (Figure 4a). With one tick, carry therefore succeeds more often than reset with twelve, on each of three training seeds (mean $\pm$ SD over seeds: $42.3 \pm 3.7\%$ vs. $36.0 \pm 2.8\%$). This is the pattern of a warm start: as in model-predictive control, one update per step suffices when each starts from the last (Section 1).

4.5 Can action chunks be both consistent and responsive?

Carry's chunks stay more consistent while nothing new is revealed, and change when the remembered target moves. Carry builds each chunk on the computation behind the previous one, so its consecutive chunks should agree while nothing new is revealed. Agreement alone could also mean ignoring new observations, so we also test whether the chunks change when the evidence does. We measure chunk disagreement, the mean distance between the translation commands that two consecutive chunks predict for the steps they share (after Bidirectional Decoding (Liu et al., 2025); Appendix D.6). It compares predicted chunks, not the executed motion.

On the five MIKASA tasks, carry's consecutive chunks disagree 36.4% less than reset's (95% interval 32.5–40.0%; Figure 5a; Appendix D.6).

Carry's chunks also change when they should. We test the same carry model on RememberColor3Move, our variant of RC3 in which one cube moves 8 cm three steps after the cubes appear, before the robot reaches any. The model never saw a cube move during training. Each of 100 scenes that carry solves is run twice: once moving the target cube (the one whose color was shown earlier) and once moving a non-target cube. Only the target move should change the plan. In all 100 scenes, carry's next chunk changes more after the target move than after the non-target move (Figure 5b). In the 61 scenes where the path it takes when nothing moves would miss the moved target, carry changes course and reaches it first in 39. Carry succeeds in 65% of scenes when the target moves and 100% when a non-target cube moves.

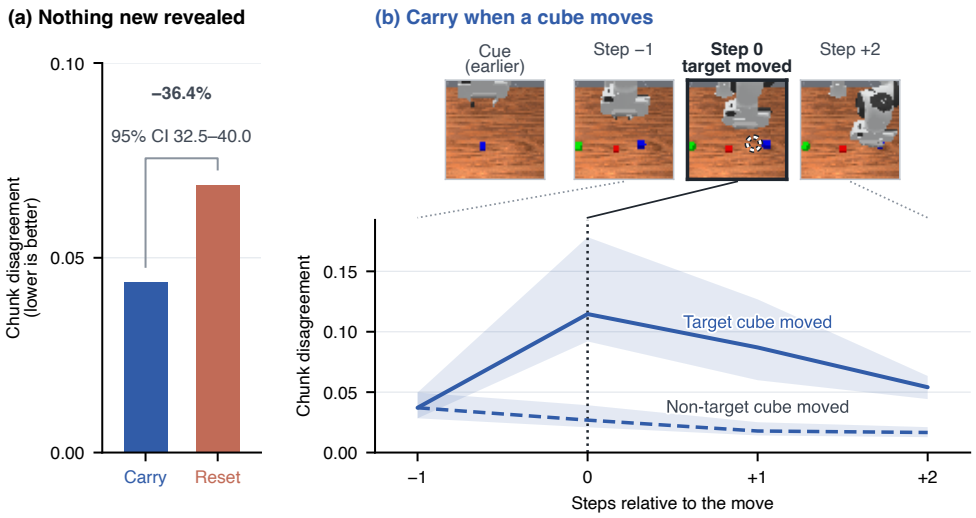


Figure 5: **Carry's chunks stay more consistent while nothing new is revealed, and change when the remembered target moves.** (a) Five MIKASA tasks, 100 paired scenes each; only steps whose actions are executed, excluding the step at which the cubes appear. Lower is better. (b) RememberColor3Move: the target cube or a non-target cube is moved 8 cm; the robot first sees the move at step 0. Lines: medians over 100 scenes; bands: 25th–75th percentiles. Details in Appendix D.6.

4.6 Does continuing help with less frequent VLM updates?

Carry maintains success with stale VLM context. By decoupling the smaller action expert from the VLM, dual-system VLAs can run the action expert at a much higher rate (Figure AI, 2025; NVIDIA, 2025). Because LycoReco is so small, this is of interest to us and a natural next step. We emulate this within the simulator by running PaliGemma every $R \in \{1, 2, 4, 8\}$ control steps, while the expert runs at every step using the most recent PaliGemma context and, through an added adapter, DINOv2/SigLIP patch features of the current frame (Appendix D.7). Neither expert loses success when PaliGemma refreshes every two or four steps (Figure 4b), so stale PaliGemma context costs little here. Carry stays above its matched reset by 5.8–9.0 points up to four steps (paired 95% intervals above zero) and by 2.3 points, within noise, at eight.

5 Discussion and Limitations

We introduced continuing computation for action experts: state is carried across decisions and refined with each new observation. Compared with matched reset experts, carry experts used earlier cues to produce different correct actions, exceeded reset's twelve-tick success with one tick per observation, and produced more consistent successive chunks while still responding when the target moved. These results show that continuing computation offers a distinct combination of memory, computation reuse, and temporal consistency. The lifetime of an action expert's state is therefore a fundamental architectural choice, alongside action representation, memory placement, and VLM–expert coupling.

LycoReco's compact expert also makes it a natural component for dual-system VLAs, while its carried state provides a separate locus for memory outside the VLM. Our experiments use simulation to isolate state lifetime under controlled and repeatable conditions, leaving how these benefits transfer to asynchronous physical execution, how memory should be divided between expert and VLM, and whether the expert can adapt its own number of ticks (Graves, 2016; Tur et al., 2026) without sacrificing reactivity or performance, to future work.

REPRODUCIBILITY STATEMENT

The appendix specifies the architecture, sequence training, complete state lifetime, baselines, and evaluation procedures. Structured result files and generation scripts support verification of the reported measurements and quantitative plots. Code, configurations, and trained checkpoints will be released upon publication.

ETHICS STATEMENT

This study uses public robot-learning datasets and simulation benchmarks. It introduces no new human-subject data. The policies are evaluated in simulation; successful simulated behavior does not establish deployment safety on physical robots.

AI USE STATEMENT

Generative AI tools assisted with literature search, manuscript drafting and review, figure and analysis code, and interpretation of experimental results, including checking consistency between claims and experimental protocols. All experimental measurements reported in this work were obtained by the authors. The authors reviewed and verified all AI-assisted content and retain full responsibility for the methodology, results, claims, and final manuscript.

REFERENCES

Vaidehi Bagaria, Bijo Sebastian, and Nirav Kumar Patel. Recursive Belief Vision Language Action Models. arXiv preprint arXiv:2602.20659, 2026. URL https://arxiv.org/abs/2602.20659.

Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bošnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier Henaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. PaliGemma: A versatile 3B VLM for transfer. arXiv preprint arXiv:2407.07726, 2024. URL https://arxiv.org/abs/2407.07726.

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*, volume 305 of *Proceedings of Machine Learning Research*, pp. 17–40. PMLR, 2025a. URL https://proceedings.mlr.press/v305/black25a.html.

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*, LosAngeles, CA, USA, June 2025b. doi: 10.15607/RSS.2025.XXI.010.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-Time Execution of Action Chunking Flow Policies. In *Advances in Neural Information Processing Systems*, volume 38, pp. 33383–33407. Curran Associates, Inc., 2025c. doi: 10.52202/085713-1122. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/300ccb2187dedd4edcc07f7e76d8e553-Abstract-Conference.html.

Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion Forcing: Next-token Prediction Meets Full-Sequence Diffusion. In *Advances in Neural Information Processing Systems*, volume 37, pp. 24081–24125. Curran Associates, Inc., 2024. doi: 10.52202/079017-0759. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/2aee1c4159e48407d68fe16ae8e6e49e-Abstract-Conference.html.

Egor Cherepanov, Nikita Kachaev, Alexey K. Kovalev, and Aleksandr I. Panov. Memory, Benchmark & Robots: A Benchmark for Solving Complex Tasks with Reinforcement Learning. In *International Conference on Learning Representations*, pp. 9759–9815, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/1088fffae5f2aa8c8324e4f45f62248d-Abstract-Conference.html.

Egor Cherepanov, Nikita Kachaev, Daniil Zelezetsky, Aydar Bulatov, Artem Pshenitsyn, Yuri Kuratov, Alexey Skrynnik, Aleksandr I. Panov, and Alexey K. Kovalev. μVLA: On Recurrent Memory for Partially Observable Manipulation in VLA Models. arXiv preprint arXiv:2606.12497, 2026b. URL https://arxiv.org/abs/2606.12497.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. In *Proceedings of Robotics: Science and Systems*, 2023. doi: 10.15607/RSS.2023.XIX.026. URL https://roboticsproceedings.org/rss19/p026.html.

Luke Darlow, Ciaran Regan, Sebastian Risi, Jeffrey Seely, and Llion Jones. Continuous Thought Machines. In *Advances in Neural Information Processing Systems*, volume 38, pp. 21548–21594. Curran Associates, Inc., 2025. doi: 10.52202/085713-0728. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/1f1628d502c62ef3725fb3b0b8eb4219-Abstract-Conference.html.

Moritz Diehl, Hans Georg Bock, and Johannes P. Schlöder. A Real-Time Iteration Scheme for Nonlinear Optimization in Optimal Feedback Control. *SIAM Journal on Control and Optimization*, 43(5):1714–1736, 2005. doi: 10.1137/S0363012902400713.

Figure AI. Helix: A Vision-Language-Action Model for Generalist Humanoid Control. https://www.figure.ai/news/helix, 2025. Technical report, February 20, 2025.

Tilmann Gneiting and Adrian E. Raftery. Strictly Proper Scoring Rules, Prediction, and Estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007. doi: 10.1198/016214506000001437.

Alex Graves. Adaptive Computation Time for Recurrent Neural Networks. arXiv preprint arXiv:1603.08983, 2016. URL https://arxiv.org/abs/1603.08983.

Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*, 2015. URL https://cdn.aaai.org/ocs/11673/11673-51288-1-PB.pdf.

Sigmund H. Høeg, Yilun Du, and Olav Egeland. Fast Policy Synthesis with Variable Noise Diffusion Models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4821–4828, 2025. doi: 10.1109/ICRA55743.2025.11127858. URL https://doi.org/10.1109/ICRA55743.2025.11127858.

Yutong Hu, Jan-Nico Zaech, Nikolay Nikolov, Yuanqi Yao, Sombit Dey, Giuliano Albanese, Renaud Detry, Luc Van Gool, and Danda Paudel. AR-VLA: Autoregressive Action Expert for Vision–Language–Action Models. In *Proceedings of Robotics: Science and Systems*, 2026. doi: 10.15607/RSS.2026.XXII.085. URL https://www.roboticsproceedings.org/rss22/p085.html.

Huiwon Jang, Sihyun Yu, Heeseung Kwon, Hojin Jeon, Younggyo Seo, and Jinwoo Shin. ContextVLA: Vision-Language-Action Model with Amortized Multi-Frame Context. arXiv preprint arXiv:2510.04246, 2025. URL https://arxiv.org/abs/2510.04246.

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*, 2025a. doi: 10.15607/RSS.2025.XXI.017. URL https://www.roboticsproceedings.org/rss21/p017.html.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Pannag 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*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 2025b. URL https://proceedings.mlr.press/v270/kim25c.html.

Myungkyu Koo, Daewon Choi, Taeyoung Kim, Kyungmin Lee, Changyeon Kim, Younggyo Seo, and Jinwoo Shin. HAMLET: Switch your Vision-Language-Action Model into a History-Aware Policy. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2510.00695.

Hao Li, Shuai Yang, Yilun Chen, Xinyi Chen, Xiaoda Yang, Yang Tian, Hanqing Wang, Tai Wang, Dahua Lin, Feng Zhao, and Jiangmiao Pang. Towards Efficient and Robust Manipulation via Multi-Frame Vision-Language-Action Modeling. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 40, pp. 18388–18396, 2026. doi: 10.1609/aaai.v40i22.38903. URL https://ojs.aaai.org/index.php/AAAI/article/view/38903.

Xinghang Li, Minghuan Liu, Hanbo Zhang, Cunjun Yu, Jie Xu, Hongtao Wu, Chilam Cheang, Ya Jing, Weinan Zhang, Huaping Liu, Hang Li, and Tao Kong. Vision-Language Foundation Models as Effective Robot Imitators. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/71639c317fb0bf398835627b4418693e-Abstract-Conference.html.

Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, Sergey Levine, Jiajun Wu, Chelsea Finn, Hao Su, Quan Vuong, and Ted Xiao. Evaluating Real-World Robot Manipulation Policies in Simulation. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 3705–3728. PMLR, 2025. URL https://proceedings.mlr.press/v270/li25c.html.

Shijie Lian, Bin Yu, Xiaopeng Lin, Laurence T. Yang, Zhaolong Shen, Changti Wu, Yuzhuo Miao, Cong Huang, and Kai Chen. LangForce: Bayesian Decomposition of Vision Language Action Models via Latent Action Queries. In *International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2601.15197.

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 (ICLR)*, 2023. URL https://openreview.net/forum?id=PqvMRDCJT9t.

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 (NeurIPS), Datasets and Benchmarks Track*, volume 36, 2023. doi: 10.52202/075280-1939. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/8c3c666820ea055a77726d66fc7d447f-Abstract-Datasets_and_Benchmarks.html.

Yuejiang Liu, Jubayer Ibn Hamid, Annie Xie, Yoonho Lee, Maximilian Du, and Chelsea Finn. Bidirectional Decoding: Improving Action Chunking via Guided Test-Time Sampling. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=qZmn2hkuzw.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What Matters in Learning from Offline Human Demonstrations for Robot Manipulation. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 1678–1690. PMLR, 2022. URL https://proceedings.mlr.press/v164/mandlekar22a.html.

NVIDIA. GR00T N1: An Open Foundation Model for Generalist Humanoid Robots. arXiv preprint arXiv:2503.14734, 2025. URL https://arxiv.org/abs/2503.14734.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An Open-Source Generalist Robot Policy. In *Proceedings of Robotics: Science and Systems*, 2024. doi: 10.15607/RSS.2024.XX.090. URL https://www.roboticsproceedings.org/rss20/p090.html.

Physical Intelligence. OpenPI: LIBERO Benchmark. https://github.com/Physical-Intelligence/openpi/tree/main/examples/libero, 2026. Repository documentation, results for the 30k-step pi05_libero checkpoint; accessed 26 September 2026.

Boyang Shen, Kaixiang Yang, Hao Wang, Qiuyu Yu, Qiang Xie, Qiang Li, and Zhiwei Wang. LoopVLA: Learning Sufficiency in Recurrent Refinement for Vision-Language-Action Models. arXiv preprint arXiv:2605.09948, 2026. URL https://arxiv.org/abs/2605.09948.

Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. MemoryVLA: Perceptual-Cognitive Memory in Vision-Language-Action Models for Robotic Manipulation. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/1fa3d6ccbcd15f7285fee666b2bd57be-Abstract-Conference.html.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics. arXiv preprint arXiv:2506.01844, 2025. URL https://arxiv.org/abs/2506.01844.

Marcel Torne, Karl Pertsch, Homer Walke, Kyle Vedder, Suraj Nair, Brian Ichter, Allen Z. Ren, Haohuan Wang, Jiaming Tang, Kyle Stachowicz, Karan Dhabalia, Michael Equi, Quan Vuong, Jost Tobias Springenberg, Sergey Levine, Chelsea Finn, and Danny Driess. MEM: Multi-Scale Embodied Memory for Vision Language Action Models. arXiv preprint arXiv:2603.03596, 2026. URL https://arxiv.org/abs/2603.03596.

Yalcin Tur, Jalal Naghiyev, Haoquan Fang, Wei-Chuan Tsai, Jiafei Duan, Dieter Fox, and Ranjay Krishna. Recurrent-Depth VLA: Implicit Test-Time Compute Scaling of Vision-Language-Action Models via Latent Iterative Reasoning. arXiv preprint arXiv:2602.07845, 2026. URL https://arxiv.org/abs/2602.07845.

Homer Walke, Kevin Black, Abraham Lee, Moo Jin Kim, Max Du, Chongyi Zheng, Tony Zhao, Philippe Hansen-Estruch, Quan Vuong, Andre He, Vivek Myers, Kuan Fang, Chelsea Finn, and Sergey Levine. BridgeData V2: A Dataset for Robot Learning at Scale. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1723–1736. PMLR, 2023. URL https://proceedings.mlr.press/v229/walke23a.html.

Chuan Wen, Jierui Lin, Trevor Darrell, Dinesh Jayaraman, and Yang Gao. Fighting Copycat Agents in Behavioral Cloning from Observation Histories. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1b113258af3968aaf3969ca67e744ff8-Abstract.html.

Chushan Zhang, Ruihan Lu, Jinguang Tong, Xuesong Li, Yikai Wang, and Hongdong Li. EvoScene-VLA: Evolving Scene Beliefs Inside the Action Decoder for Chunked Robot Control. arXiv preprint arXiv:2605.21862, 2026. URL https://arxiv.org/abs/2605.21862.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Robotics: Science and Systems (RSS)*, 2023. doi: 10.15607/RSS.2023.XIX.016. URL https://www.roboticsproceedings.org/rss19/p016.html.

A ARCHITECTURE AND STATE LIFETIME

The expert uses a single CTM with D neurons for the complete action chunk. Table 3 gives its architecture, which we hold the same on all three benchmarks; Appendix B lists benchmark-specific settings.

The expert reads PaliGemma's final-layer keys and values directly, without an image/text projection or context layer normalization. A linear map Linear(d_p, D) of the d_p-dimensional proprioception, followed by LayerNorm(D), encodes it as a token of width D; local key/value projections append this proprioception token and a token formed from the current post-activations z. Four query heads of width 256 share the VLM's key/value head. Images enter the expert through the VLM only, except in the dual-rate study (Appendix D.7). PaliGemma receives the instruction and the benchmark's RGB cameras, resized to 224×224: the third-person and wrist cameras on LIBERO, one camera on SIMPLER-WidowX, and the top and wrist cameras on MIKASA. Proprioception has $d_p = 8$ on LIBERO and 7 on SIMPLER-WidowX and MIKASA, and actions have $d_a = 7$. The complete systems have 2.92B VLM parameters and an expert of about 3M; joint training updates every VLM parameter upstream of the final-layer key/value readout (Appendix B).

Table 3: CTM expert architecture. Parameter counts include adapters and the chunk readout.

Component	Configuration
Neurons	$D = 512$
Pre-activation history / private hidden width	$M = 32$ / 16
Synapse network	Depth-two CTM U-Net, input width $2D$
Cross-attention	Four heads, one query, no dropout
Attention / output synchronization	1024 neuron pairs each
Pair selection	Random pairs with replacement, fixed after initialization
Readout	Linear, $1024 \to Hd_a$ (chunk length H, action dimension d_a)
Expert parameters / carried state	3.0–3.1M (varies with H) / 82 KiB FP32

State transition and readout. Dropping the step index t, the state after tick k is $h_k = (z_k, U_k, \alpha_k^{\text{att}}, \beta_k^{\text{att}}, \alpha_k^{\text{out}}, \beta_k^{\text{out}})$, with post-activations $z \in \mathbb{R}^D$, pre-activation history $U \in \mathbb{R}^{D \times M}$, and one accumulator pair for each selected neuron pair in the attention and output sets. Tick k first updates the attention set with z_{k-1} by Eq. (2) and then computes

$$\begin{aligned} v_k &= \text{Attn}\big(W_q S_k^{\text{att}},\ \text{KV}(o)\big), \qquad u_k = f_{\text{syn}}([v_k; z_{k-1}]), \\ U_k &= \text{append}_M(U_{k-1}, u_k), \qquad z_{i,k} = g_i(U_{i,k}), \end{aligned} \tag{3}$$

after which the output set is updated with z_k. Here $\text{KV}(o)$ denotes PaliGemma's final-layer keys and values for the current observation together with the proprioception token and a token formed from z_{k-1}, f_{syn} is the shared synapse network, and g_i is neuron i's private network. After T ticks the readout is

$$\hat{A} = \text{reshape}_{H \times d_a}\big(W_o S_T^{\text{out}} + b_o\big). \tag{4}$$

The horizon expands only at readout: the state is not a queue of action slots shifted after execution. For carry, the state after T ticks is also the state the next observation's ticks start from, and an old accumulator then contributes $r_{ij}^T \alpha_{ij}$; the tick budget therefore affects both the current prediction and what is carried forward.

Transition networks. The synapse network first maps $2D \to D$ with a linear layer, layer normalization, and SiLU. Its down path applies Linear($D, 16$), LayerNorm(16), and SiLU; its up path applies Linear($16, D$), LayerNorm(D), and SiLU. The up-path result is added to the initial D-wide projection and layer normalized to produce u_k. Each private neuron network maps $M \to 32 \to 16 \to 2 \to 1$: independent affine maps produce 32 and two channels, and gated linear units halve those widths.

Initialization and synchronization. Post-activations and pre-activation histories have learned initial values. Pair decay rates λ_{ij} are initialized to zero, projected into $[0, 15]$, and mapped to $r_{ij} = \exp(-\lambda_{ij})$. Initial retention is therefore one, as in the CTM implementation. At a set's first

update, $\alpha_{ij} = z_i z_j$ and $\beta_{ij} = 1$. The output set is initialized from the initial post-activations; the attention set starts with its first update in the first tick. Both accumulators are retained together, using the normalization in Eq. (2). At $r_{ij} = 1$, a constant pair product c produces a feature proportional to $c\sqrt{n}$ after n updates. Learned retention changes this scale and timescale. With $M = 32$ and $T = 12$, the explicit pre-activation history spans about 2.7 observations. Longer dependencies must survive in the post-activations or the synchronization accumulators.

B Training and benchmark protocols

Objective and initialization. Let $m_{t,j,d}$ identify a valid future action coordinate, and let $\widetilde{A}$ denote normalized actions. The supervised objective is

$$\mathcal{L} = \frac{\sum_{t,j,d} m_{t,j,d} |\widehat{\widetilde{A}}_{t,j,d} - \widetilde{A}^*_{t,j,d}|}{\sum_{t,j,d} m_{t,j,d}}. \tag{5}$$

This is a global valid-coordinate mean. The encoder initializes from PaliGemma MIX-224. Joint training updates the vision and language weights upstream of the final-layer key/value readout, together with the expert. Frozen experiments hold the entire encoder fixed and train only the expert, proprioceptive adapter, and readout.

System training and selection. Each benchmark has its own checkpoint and action normalization. Robot trajectories are split by episode, with ten percent reserved for development; normalization uses training episodes only. Carry training uses chronological streams: gradients connect the observations of a window, whose length Table 4 gives, and state values carry across windows with gradients detached. No ground-truth action is fed back as input. Episode-consistent image augmentation prevents artificial frame-to-frame appearance changes within a training window. AdamW uses betas $(0.9, 0.999)$, epsilon 10^{-8}, and global gradient clipping at one. The system runs use their benchmark-specific optimization budgets in Table 4. On LIBERO and SIMPLER-WidowX, stage two (below) trains for up to 30k updates. As in OpenVLA-OFT, we test its checkpoints on the benchmark episodes and report the best performance. The MIKASA system and its matched reset expert (Appendix D.6) are the final checkpoints after 10k updates. System experts train and are evaluated at twelve ticks per observation.

Two-stage training on LIBERO and SIMPLER-WidowX. PaliGemma was never trained on robot data, so on these benchmarks we adapt it first and train the carry expert second. In stage one, PaliGemma is fine-tuned jointly with a reset expert on randomly shuffled observations. Shuffling matters for this stage: the observations of a chronological window are highly correlated, so batches of windows adapt the VLM poorly. Stage one runs for 15k updates on each benchmark, with learning rates of 10^{-5} for PaliGemma and 10^{-4} for the expert (constant on LIBERO, cosine over a 50k-update horizon on BridgeData V2). In stage two, PaliGemma is frozen and a carry expert is trained from a fresh initialization on chronological streams. Its state is carried across windows and reset at episode boundaries. Only the expert and its proprioceptive input layers receive gradients, so this stage costs far less than training the VLM with it. Stage two uses AdamW at a constant learning rate of 10^{-4} with weight decay 10^{-4}, and it zeroes the proprioceptive input on 80% of training observations. Evaluation always provides it.

LIBERO. A single policy trains on the joint forty-task pool comprising Spatial, Object, Goal, and Long. We evaluate fifty reference initial states per task, for 2,000 episodes per seed, and average the four equal-sized suites. The main comparison uses $H = 20$ and $K = 10$. Suite horizons are 280, 280, 300, and 520 environment steps.

BridgeData V2 → SIMPLER-WidowX. SIMPLER-WidowX policies train on BridgeData V2 and are evaluated on the four WidowX tasks in SIMPLER: spoon on towel, carrot on plate, stacking blocks, and eggplant in basket (Li et al., 2025; Walke et al., 2023). We evaluate 288 episodes per seed across the four tasks. We report per-task success and their unweighted mean. The main evaluation uses $H = 16$, $K = 4$ and reports native final-step success.

Table 4: Training budgets. Window: consecutive observations per stream joined by gradients. Batch: observations per update. Updates: training budget. On LIBERO and BridgeData V2, stage one trains PaliGemma with a reset expert on shuffled observations, stage two trains the carry expert with PaliGemma frozen, and the reported stage-two checkpoint is the best one evaluated, which may precede the last.

Dataset	Window	Batch	Updates	Expert LR	VLM LR	Seeds
Complete systems (Table 1)						
LIBERO, stage one	1	256	15000	10^{-4}	10^{-5}	1
LIBERO, stage two	16	1024	30000	10^{-4}	0	1
BridgeData V2, stage one	1	256	15000	10^{-4}	10^{-5}	1
BridgeData V2, stage two	16	1024	30000	10^{-4}	0	1
MIKASA	16	256	10000	2×10^{-4}	2×10^{-5}	1
Reset/carry comparisons (PaliGemma frozen)						
MIKASA	8	256	10000	2×10^{-4}	0	1
LIBERO	8	256	20000	10^{-4}	0	1

MIKASA memory evaluation. The benchmark comparison uses ShellGameTouch, InterceptMedium, and RememberColor-3/5/9, with one LycoReco model trained on the five-task mixture; the reset/carry pairs of Appendix D train on the same mixture. These are the tasks used in the reported MemoryVLA comparisons, not the entire MIKASA-Robo-VLA suite. We train and evaluate on the MIKASA-Robo-VLA releases of these tasks (the `-VLA-v0` environments and their demonstrations); the published rows in Table 1 were obtained on the original MIKASA-Robo releases (`-v0`). The releases keep each task's goal but differ in timing, in whether the robot is held still during the cue, and in demonstrations. Evaluation uses 100 native episodes per task with seeds 4242424242–4242424341, the same for every task and checkpoint, $H = 4$, and $K = 1$; the state resets between episodes. Episodes run for at most 30, 60, 25, 25, and 25 environment steps on ShellGameTouch, InterceptMedium, and RememberColor-3/5/9, under the native termination rules. We report each task and their unweighted mean. Cue and delay observations are processed even when the environment constrains robot actions. The native `success_once` predicate, success at any step of the episode, determines task completion; privileged target identity is used only for scoring diagnostics. MemoryVLA's released evaluation also scores `success_once` over 100 episodes, but with longer budgets (90, 90, 60, 60, and 60 steps), four actions per query, and seeds 1–100 (Shi et al., 2026). The OpenVLA-OFT, π_0, and Octo rows report 100-episode averages without released evaluation code; we assume the same metric.

C Complete-system comparison

Table 1 asks whether LycoReco is competitive with published VLAs. These differ from it in backbone, data, action objective, training budget and evaluation protocol, so the table does not attribute a difference to the state lifetime; Sections 4.3–4.6 test that with matched experts (Appendix D).

C.1 Reference systems and attribution

Success rates of external systems in Table 1 are published results. LIBERO means use the same four named suites; the five-suite average including LIBERO-90 is not substituted. MIKASA means use the five tasks listed above. MIKASA-Robo reports per-task success only, so the means of its Octo, OpenVLA-OFT, and π_0 rows are our unweighted averages. We keep each source's reporting precision and give no intervals for published numbers.

Published RD-VLA references. The LIBERO row is the fixed twelve-iteration result in Table I of RD-VLA (Tur et al., 2026). We time its four released suite-specific checkpoints with their published inference settings ($H = 8$, $K = 5$); re-evaluated on 50 reference starts per task, they succeed in 94.5% of episodes, against the published 93.0%.

Published MemoryVLA references. All three MemoryVLA (Shi et al., 2026) rows are its own reported results: SIMPLER-WidowX from its Table 1, MIKASA from its Table 4, and LIBERO from its Appendix J, Table 17(b), which averages the four suites used here. Its Table 3 headline also includes LIBERO-90. Its Table 4 also lists the π_0 and OpenVLA-OFT MIKASA baselines, which we cite at their source, MIKASA-Robo (Cherepanov et al., 2026a). Appendix B compares its MIKASA evaluation with ours. We did not re-evaluate its success rates; its latency and parameter counts are our measurements of its released checkpoints.

Published π_0 and $\pi_{0.5}$ references. LIBERO uses the official 30k-step `pi05_libero` result (Physical Intelligence, 2026), whose reported 96.85% mean is rounded to 96.9% in Table 1. The SIMPLER-WidowX row comes from Table 1 of LangForce (Lian et al., 2026). The SIMPLER source does not identify that baseline's checkpoint and complete training/evaluation protocol. We found no published $\pi_{0.5}$ result on the five simulated MIKASA tasks; MIKASA-Robo (Cherepanov et al., 2026a) fine-tunes $\pi_{0.5}$ only for its real-robot experiments. For MIKASA, we therefore include the four-action-chunk π_0 baseline in its Table 4, a different model from the $\pi_{0.5}$ rows on LIBERO and SIMPLER. We timed the released `pi05_libero` checkpoint in its native OpenPI runtime; the SIMPLER-WidowX $\pi_{0.5}$ and MIKASA π_0 checkpoints are unreleased.

Published OpenVLA-OFT references. LIBERO uses Table I of OpenVLA-OFT (Kim et al., 2025a): four suite-specific policies with third-person and wrist images and proprioception, $H = K = 8$, and 500 trials per suite. The authors' single policy for all four suites reports 96.8% (their Table XIV). The authors released no SIMPLER-WidowX or MIKASA policy. The SIMPLER-WidowX row comes from Table 1 of CronusVLA (Li et al., 2026), which evaluates a separate WidowX checkpoint but does not identify it or its training recipe. The MIKASA row is the eight-action-chunk OpenVLA baseline in Table 4 of MIKASA-Robo (Cherepanov et al., 2026a), fine-tuned with LoRA on 250 demonstrations per task, with action chunks and an L1 loss as in OpenVLA-OFT; MemoryVLA's Table 4 lists the same scores under that name. We timed the released LIBERO-Spatial policy in the authors' evaluation code; the other three suite policies have identical tensor shapes and code.

Published HAMLET, ContextVLA, and Octo references. HAMLET (Koo et al., 2026) reports LIBERO on GR00T N1.5 (its Table 2, one policy for all four suites) and SIMPLER-WidowX on CogACT (its Table 3, success columns). The ContextVLA (Jang et al., 2025) rows use its eight-frame π_0-FAST variant (its Tables 1 and 2), so one base model covers both benchmarks; its GR00T N1.5 variant reaches 97.0% on LIBERO but 31.8% on SIMPLER-WidowX. The Octo (Octo Model Team et al., 2024) MIKASA row is Octo-Small in Table 4 of MIKASA-Robo (Cherepanov et al., 2026a), with a ten-frame context, four-action chunks, and only the readout heads fine-tuned. Of these, only HAMLET's LIBERO policy has public weights: its first author hosts a GR00T N1.5 checkpoint that matches the paper's description but is not linked from it. We timed it in the released code. HAMLET's CogACT code, the ContextVLA fine-tunes, and MIKASA-Robo's Octo fine-tune are unreleased.

Published AR-VLA references. The SIMPLER-WidowX row is the BridgeData V2 result in Table I of AR-VLA (Hu et al., 2026), whose released checkpoint we timed.

Native baseline costs. Table 5 separates VLM (B) from action expert (M). The latter counts action-generation modules outside the VLM, including robot-state projections and memory retrieval/fusion. Costs exclude the vision/language backbone from M, but include its full forward pass in ms.

We retain each released runtime's precision: RD-VLA is BF16 throughout. MemoryVLA's SIMPLER-WidowX timing includes its native seven-query adaptive action ensemble; the MIKASA profile uses its released $H = 4$ configuration. AR-VLA runs its released FP32 weights and unmodified code.

C.2 Computation and latency accounting

Timing setup. All timings use batch one on a single RTX 3090, with 100 warm-up and 1,000 timed calls; we report the median.

Table 5: **Complete-system parameter counts.** VLM (B) counts the vision–language backbone in billions; action expert (M) counts parameters outside the VLM in millions, including robot-state adapters and memory. OpenVLA-OFT counts come from its released LIBERO policies.

Method	VLM (B)	Action expert (M)
LIBERO: one 40-task LycoReco policy		
RD-VLA (Tur et al., 2026)	1.25	73.1
MemoryVLA (Shi et al., 2026)	7.54	835.5
HAMLET (Koo et al., 2026)	1.66	1203.0
OpenVLA-OFT (Kim et al., 2025a)	7.54	167.9
$\pi_{0.5}$ (Black et al., 2025a)	2.92	430.1
LycoReco	2.92	3.1
BridgeData V2 → SIMPLER-WidowX		
MemoryVLA (Shi et al., 2026)	7.54	835.5
AR-VLA (Hu et al., 2026)	2.92	311.5
LycoReco	2.92	3.1
MIKASA: five memory tasks, one LycoReco checkpoint		
MemoryVLA (Shi et al., 2026)	7.54	835.5
LycoReco	2.92	3.0

Full-policy latency. Our full-policy timing measurements cover complete decisions over a common ten-observation LIBERO trace. Each call refreshes perception and returns a CPU action ($K = 1$ for timing), including native preprocessing, input transfers, state updates, and output conversion. This is fresh-decision cost, not amortized chunk cost or autonomous-rollout latency. Carry processes every control observation (Section 3.2), so a carry system that executes K actions per chunk makes K such calls per chunk, where reset makes one; its per-chunk expert cost in Table 6 grows by the same factor. LycoReco is timed in each benchmark's configuration, with the fine-tuned VLM (FP32 weights, BF16 computation) and $T = 12$ ticks: two camera views and $H = 20$ on LIBERO, one view and $H = 16$ on SIMPLER-WidowX, and two views and $H = 4$ on MIKASA.

Action-expert computation. We count two FLOPs per multiply–add in matrix products and attention, the convention of PyTorch's FlopCounterMode; for LycoReco's LIBERO expert, our count matches FlopCounterMode exactly. An expert's cost per chunk is roughly twice its parameter count, times the number of vectors it processes per pass, times the number of passes. A flow-matching expert processes all H action tokens at each of its N integration steps, and a direct L1 head, as in OpenVLA-OFT (Kim et al., 2025a), processes them once. A CTM tick processes one vector, whatever H. Table 6 holds the parameter count at that of $\pi_{0.5}$'s LIBERO expert ($H = 10$, $N = 10$). Widened to $D = 9{,}896$ neurons, our CTM needs 10.3 GFLOPs per chunk, about the cost of one of $\pi_{0.5}$'s flow steps (7.91 GFLOPs), against 79.1 GFLOPs for all ten. The VLM dominates both complete policies: a query costs 3.04 TFLOPs for LycoReco (two views, 640 tokens) and 4.71 TFLOPs for $\pi_{0.5}$ (three image slots, 968 tokens).

Table 6: **Action-expert computation per chunk.** GFLOPs per pass and per chunk, excluding the VLM. The first three rows share $\pi_{0.5}$'s LIBERO setting ($H = 10$, a 968-token prefix); direct L1 omits the time conditioning (118.6M parameters) that only flow needs, and the widened CTM is counted, not trained. The last row is LycoReco in its own LIBERO setting.

Chunk generation	Params (M)	Passes	GFLOPs/pass	GFLOPs/chunk
Flow matching ($\pi_{0.5}$ expert)	430.1	$N = 10$	7.91	79.1
Direct L1 (same network, one pass)	311.5	1	7.67	7.67
CTM ticks, $D = 9{,}896$	430.1	$T = 12$	0.860	10.3
LycoReco, $D = 512$, $H = 20$	3.1	$T = 12$	0.00846	0.102

D RESET/CARRY COMPARISONS

Every reset/carry comparison in Section 4 pairs experts trained with the same recipe, specified below. Apart from two budget-sweep cells with three seeds (Appendix D.5), each uses one training seed per expert, so its intervals cover evaluation noise, not training variance.

D.1 PAIRED TRAINING

What is shared and what differs. Wherever reset and carry experts are compared, they share the PaliGemma encoder (frozen, except that the pair of Sections 4.3 and 4.5 fine-tunes it with the expert), data split, action normalization, image augmentation (none on LIBERO's cached features), optimizer, learning-rate schedule, number of updates, and training seeds. Carry experts train on chronological streams: each update draws a fixed number of streams, each a window of consecutive observations (Table 4); gradients connect the observations of a window, and state values continue across windows with gradients detached. Reset returns its entire state to the learned initial state at every observation, in training and evaluation. The MIKASA and budget-study reset and carry experts train on the same chronological windows, so both see identical samples. Neither expert receives past images or an explicit history bank. Future observations, demonstrated actions, cue labels, rewards, and target positions never enter the policy. An evaluation-only lesion is never substituted for a trained reset expert.

Tick budgets. Every expert trains and is evaluated at one fixed tick budget, so its state is never evaluated under a budget it was not trained with.

MIKASA experts. Sections 4.3 and 4.5 use the MIKASA model of Table 1 and a reset expert trained by its recipe (Table 4). Unless noted, the other MIKASA comparisons below use a carry and reset pair on frozen features, the lower rows of Table 2. Both are the CTM of Table 3 with $H = 4$ on frozen PaliGemma MIX-224 features, trained for 10k updates at batch 256 with episode-consistent image augmentation. Carry trains on 32 chronological streams of eight frames per update, with gradients inside each window, the state carried across windows, and every frame supervised. Each task's last 25 demonstrations are excluded from training targets, giving 1,125 training and 125 held-out episodes. For these frozen experts, normalization statistics use the full corpus, so the held-out episodes are diagnostic examples with shared corpus statistics, not a fully untouched validation protocol. Evaluation uses deterministic image processing and one executed action per observation. The reset + full history row of Table 2 is the reset-with-history expert of Appendix E; its 64-frame bank is longer than any episode of the five tasks (at most 60 steps), so it holds every earlier frame.

D.2 PAIRED CUES (SECTION 4.3)

Fifty RC3 scenes are reconstructed three times, once per cue color, with identical decision images, proprioception, instruction, and cube layout, which both experts receive. The native wrapper freezes actions before the decision onset, the first step at which the robot may act; the expert still processes those observations and then acts with $K = 1$ within the native 25-step horizon. First reach is the first cube within 0.05 m of the tool center point; completion uses the native success predicate, which also requires the robot to be static. Target identity is used only for scoring.

Paired-history control. A fourth replay of each scene repeats the first cue and reproduces its actions and outcomes exactly. On ordinary episodes of the same 50 scenes, carry succeeds in 44/50 and reset in 8/50, against 133/150 and 50/150 replays; the scenes are harder than the benchmark's, on which the same experts reach 93% and 31% (Table 1 and Appendix D.6).

Selected frames. Figure 3 uses the paired-cue scene with the smallest seed among those with all three carry successes and exactly one reset success, and shows the top camera without modifying its pixels.

Scene-specific reset choices. Across the 50 RC3 scenes, the reset expert first reaches red in 22, lime in 13, and blue in 15, and it reaches a cube in every replay. Within every scene its decision action, its later actions, and its first-reached target are invariant across cues; it completes 50 of the

Table 7: **Parts of the carried CTM state (RC3 paired cues).** Frozen-feature experts, not the MIKASA model of Table 1: the intact and reset-once rows use the carry expert of Table 2 (lower rows). One-time resets affect only the selected components at decision onset. The activation-only control is trained with the same frozen-feature recipe to carry only post-activations.

Carry expert	Completed (/150)
Intact	119
Neuron histories reset once	55
Synchronization reset once	26
Both reset once	33
Activation-only (trained)	64

150 replays, one per scene. The selected red reach is therefore a scene-specific response to identical current inputs, not a universal red preference.

D.3 Parts of the carried CTM state (Section 4.3)

This ablation asks which parts of the carried CTM state (post-activations, per-neuron pre-activation histories, or synchronization accumulators) the trained carry expert relies on, and whether the histories and synchronization it carries beyond its post-activations help.

Protocol. One-time resets use the frozen-feature carry expert (Table 2, lower rows) and the paired-cue scenes. Immediately before the first decision prediction, the selected component is replaced by its initial value. Neuron histories are replaced by the learned initial history, and the attention-set accumulators by zero numerator and denominator, equivalent to their uninitialized state at the next update. The output-set accumulators are replaced by the retained post-activations' pair products with denominator one. Post-activations and unselected components are unchanged, and all later updates proceed normally. Such resets measure what the trained expert relies on, not what a model trained without the component could achieve. The activation-only expert is trained with the frozen-feature MIKASA recipe above; it carries post-activations across observations but restarts histories and synchronization at every observation.

D.4 State swaps (Section 4.3)

Table 8: **Whole-state swaps (RC3).** The same 50 recipients in every row, each with one paired donor scene. The predicted cue comes from the action-choice readout, not from completed reaches.

State	Predicted cue	Trials
Own	Recipient	47/50
Same cue	Recipient	38/50
Different cue	Donor	37/50
Different cue	Recipient	6/50

Protocol. The swaps reuse the saved decision-onset prefixes of the 50 RC3 scenes of the frozen-feature carry expert's paired cues (Table 2, lower rows), without new simulator rollouts. Each scene is a recipient with one of its cues (17, 17 and 16 per color). Its donor is another scene at the same decision-onset age with the nearest proprioception, which supplies two states: one with the recipient's cue and one with a different cue. Donors are fixed before any prediction.

The entire state entering the decision is replaced while RGB, proprioception, and instruction stay fixed. The predicted cue comes from an action-choice readout: a ridge map, fit on the 225 RC3 training demonstrations, from top-camera cube centroids to the summed planar motion of the demonstrated chunk; a predicted chunk is assigned to the visible cube whose mapped motion is nearest. Applied to held-out demonstrations, it recovers the demonstrated cube from the demonstrated actions in 25/25 cases.

Result. Different-cue donors move the predicted choice to the donor's cue in 37/50 recipients, and only 6/50 keep their own cue; same-cue donors preserve it in 38/50. The own-state control keeps 47/50, so transplanting a state also disrupts some choices. No completion is measured after a swap.

D.5 Budget sweep (Section 4.4)

Experts and evaluation. For each $T \in \{1, 2, 4, 8, 12\}$ and CTM state lifetime, we train one expert per seed on frozen PaliGemma MIX-224 final-layer keys and values for 20k updates (AdamW, weight decay 10^{-4}), with reset and carry on the same chronological windows, 32 streams of eight observations per update. Each expert runs from episode start at its training budget on the 40 LIBERO tasks with ten initial states each (400 episodes per expert). It predicts $H = 20$ actions and executes $K = 1$ per decision.

Seeds. Each cell of Table 9 has one training seed, so we report success rates without intervals. The exceptions are the two cells behind the comparison in Section 4.4, carry at one tick and reset at twelve, which have two more seeds that set the initial weights and the data order (lower block of Table 9). Across seeds, success varies with a standard deviation of 2.8 to 3.7 points, so single-seed differences of that size are within training noise.

Table 9: **Success by tick budget (LIBERO).** Success (%) over 400 episodes per expert, with $K = 1$. Upper block: one training seed per separately trained expert, as plotted in Figure 4a. Lower block: selected cells across three training seeds, with mean ± standard deviation.

Lifetime	$T=1$	$T=2$	$T=4$	$T=8$	$T=12$
Reset	11.5	16.3	30.0	34.5	38.8
Carry	46.5	44.8	47.8	49.8	50.3

Cell	Seed 1000	Seed 1001	Seed 1002	Mean ± SD
Carry, $T=1$	46.5	39.8	40.5	42.3 ± 3.7
Reset, $T=12$	38.8	33.3	36.0	36.0 ± 2.8

D.6 Chunk disagreement (Section 4.5)

Equal-weighted chunk disagreement. Let $\hat{a}_{t,j}$ be the unnormalized, unclipped action predicted at step t for execution at step $t + j$, and $\hat{a}^{\text{xyz}}_{t,j}$ its translation command. Equal-weighted chunk disagreement averages the distance between consecutive chunks over the three steps they share, $\Delta_t = \frac{1}{3}\sum_{j=0}^{2} \|\hat{a}^{\text{xyz}}_{t,j} - \hat{a}^{\text{xyz}}_{t-1,j+1}\|_2$. This is adapted from the backward-coherence criterion of Liu et al. (2025), using translation coordinates and an equal-weight average over shared steps.

Experts. Figure 5a compares the MIKASA model of Table 1, which carries its full CTM state until the episode boundary, with a reset expert trained by the same recipe: the same initialization, demonstrations, 16 chronological streams of 16 observations, joint fine-tuning of PaliGemma and the expert, optimizer, schedule, and 10,000 updates. The only difference is that reset returns its entire state to the learned initial state before every observation, in training and evaluation. Both run at $T = 12$, $H = 4$, and $K = 1$, and PaliGemma recomputes its keys and values at every observation.

Scenes. Both experts run the 100 evaluation scenes of Appendix B on each task. Scene seeds, targets, phase durations, and initial geometry are paired; observations can diverge after different actions. Of these 500 episodes, carry succeeds in 313, matching Table 1, and reset in 129: 30, 41, 31, 15, and 12 on ShellGameTouch, InterceptMedium, and RememberColor3/5/9.

Counted steps and result. Panel (a) applies the Δ_t formula above; a step counts only if the three actions it compares are executed. An episode ends at its first success, or else at its step limit, so its last two steps are excluded. During the ShellGameTouch and RememberColor cue and delay, the benchmark holds the robot still and discards the policy's actions, so counting starts when manipulation begins. We also exclude the RememberColor layout reveal, the step at which the cubes

appear and new evidence arrives. Each episode contributes its median, and task means are averaged with equal weights. Carry's mean episode-median Δ_t is 43.7 and reset's is 68.7, in 10^{-3} controller units, a 36.4% reduction (95% interval 32.5–40.0%, resampling paired scenes within task); counting the reveal step as well gives 35.0%. The reduction differs by task: 48.9% on InterceptMedium, 48.5% on RememberColor3, 46.7% on RememberColor5, 26.6% on ShellGameTouch, and 14.7% on RememberColor9. Chunk disagreement is not executed-command variation or physical jerk, and a consistent chunk need not be a correct one.

Moved cube. RememberColor3Move is our variant of RememberColor3: one cube moves during the episode, and the step limit is 60 instead of the native 25, so that the robot has time to re-approach a moved cube. The test is zero-shot: no training episode moves a cube. The expert is the carry model of Figure 5a. Reset is not tested: it never knows which cube is the target, so it cannot show whether a policy that knows the target clings to its plan. Scenes come from seeds disjoint from the benchmark's, taken in order until 100 qualify (333 checked). To qualify, carry must succeed within 60 control steps with nothing moved. Because the scenes were first drawn for a comparison with a frozen-feature reset expert, that expert must also succeed with nothing moved. The move must also be visible in the policy cameras, settle within 3 mm, keep cubes at least 6 cm apart, and happen before the robot reaches any cube. Each scene runs the two variants of Section 4.5, the target cube moved and a non-target cube moved, plus a run with nothing moved that gives the path the robot takes when nothing moves; all three use identical simulator and policy seeds. The move translates the cube by 8 cm, in a direction that is a fixed function of the layout; a larger move could take the cube out of the cameras' view and would make the task much harder. The first changed observation comes three steps after the layout reveal. Observations and chunks before the move are identical across the three runs. Figure 5b applies the Δ_t of panel (a) to both variants. An episode ends at its first success, and Δ_t counts only while its three shared steps are executed; every run is still in progress two steps after the move. The thumbnails are the carry expert's saved top-camera inputs for the first scene in which carry, with the target moved, succeeds without entering the target's old position. At the first changed observation, carry's median Δ_t is 114.6 when the target moves and 26.8 when a non-target cube moves (26.7 when nothing moves; 10^{-3} controller units). Success is the native criterion, the tool center point within 5 cm of the target with the robot static, within 60 control steps. When the target moves, the path the robot takes when nothing moves would still reach the moved target first if its tool positions, checked against the moved layout, come within 5 cm of the target before any other cube. A change of course is counted when carry's first cube is the target and that path would not reach it first.

D.7 Dual-rate execution (Section 4.6)

The dual-rate study emulates the rate separation of dual-system VLAs in the synchronous simulator. PaliGemma recomputes keys and values only every R control steps, starting at the first step of the episode. Between refreshes, the expert reads the most recent keys and values together with the current proprioception and the current frame's image patches, updates at every step with $T = 12$ ticks, and executes one action per step. The dual-rate experts are 2.5k-update fine-tunes of trained $T = 12$ experts on frozen PaliGemma features (128 streams of two observations, 10k updates): carry is fine-tuned from the carry expert, and reset from the reset expert, which starts every observation from the initial state. The fine-tune adds an OpenVLA-style patch adapter to the action expert (Kim et al., 2025b): the current frame's DINOv2 and SigLIP patch features are concatenated per patch, read by cross-attention at every tick and added to the read of the VLM keys and values. This auxiliary patch read does not refresh PaliGemma's cached keys and values, which remain stale until the next VLM update. The adapter's output projection starts at zero, so each fine-tune starts from its source expert's actions exactly. PaliGemma, SigLIP and DINOv2 stay frozen; only the adapter and the expert train, at a constant learning rate of 10^{-4}, with the same data order for both lifetimes. Each training episode samples R uniformly from $\{1, 2, 4, 8\}$ and supplies the context of the most recent refresh. Both models see the same staleness distribution. Each expert is evaluated at every R on the 40 LIBERO tasks with ten initial states each (400 episodes, the same initial states in every cell), with 95% intervals that resample initial states within each task. The simulator still waits for inference, so the study measures behavior under stale context, not concurrency or inference latency.

E Recurrent Transformer controls

The five-task comparison trains CTM and a recurrent Transformer (RT) each as a carry and a reset expert. RT is our own implementation of the same state lifetimes, not a reproduction of RD-VLA or AR-VLA. All four experts use the MIKASA recipe of Appendix D.1 ($H = 4$, $K = 1$, $T = 12$, 10k updates on frozen PaliGemma features) and are paired as in Appendix D.

The carry RT retains eight 512-dimensional latent tokens across observations. Its shared block uses four-head self-attention, cross-attention to the final-layer K/V and a width-2048 feed-forward layer, repeated twelve times. Fresh action-query tokens predict the four actions. The reset RT starts training from the same untrained weights and block, but replaces its latent tokens with the learned initial tokens before each observation, in training and in evaluation. The CTM experts are the frozen-feature pair of Appendix D.1. Each RT has 4,482,055 trainable parameters, and the carry RT retains 16 KiB of FP32 expert state, versus 2,995,790 parameters and 82 KiB for CTM; neither parameter count nor state size is matched.

Table 10: **Carry and reset for CTM and RT (MIKASA).** Native success (%) over 100 episodes per task, with the same scene seeds for every expert. Reset + full history also reads all earlier top-camera VLM features.

Expert	Lifetime	SGT	IM	RC3	RC5	RC9	Mean
CTM	carry	61.0	24.0	89.0	52.0	12.0	47.6
CTM	reset	22.0	22.0	30.0	17.0	11.0	20.4
CTM	reset + full history	65.0	33.0	57.0	26.0	11.0	38.4
RT	carry	45.0	22.0	66.0	30.0	11.0	34.8
RT	reset	15.0	21.0	32.0	22.0	10.0	20.0

Because all four experts keep PaliGemma frozen, they score below the LycoReco system of Table 1, which fine-tunes PaliGemma with its expert (MIKASA mean 62.6% against 47.6% for the CTM carry expert; Section 4.1). Carrying state raises the CTM mean by 27.2 percentage points (paired 95% interval $[21.8, 32.6]$) and the RT mean by 14.8 ($[9.6, 20.0]$). Both gains come mainly from ShellGameTouch and RememberColor3, and the CTM gain also from RememberColor5; on InterceptMedium and RememberColor9 all four experts lie within three points of each other. The two reset experts are nearly equal (0.4 points apart, $[-4.4, 5.2]$). Nearly all of the carry CTM–RT difference of 12.8 points ($[7.2, 18.2]$) therefore comes from what carrying adds. CTM gains 12.4 points more from it than RT does ($[5.0, 19.8]$), mostly on RememberColor3 and RememberColor5 rather than uniformly across tasks. These native episodes are separate from the paired-cue replays of Appendix D.2. Intervals resample the 100 scene seeds, keeping a seed's five tasks together for means.

Reset with full history. A reset expert that can see the past tests whether carry's advantage is only access to earlier observations. This CTM expert restarts from its initial state at every observation, like reset, but on every tick it also attends to the frozen PaliGemma top-camera keys and values of every earlier frame of the episode (its 64-frame bank is longer than any MIKASA episode), with a learned frame-age embedding (1.07M extra parameters). It is trained exactly like the CTM reset expert, from the same initializer with the new read's output map at zero, so it starts as that expert. It reaches a mean of 38.4%, between reset and carry: it matches or beats carry on ShellGameTouch and InterceptMedium but stays well below it on RememberColor3 and RememberColor5, where a cue must be recalled after a delay. With its history emptied at every step it falls to 9.4%, so it does use the past frames. Seeing the past frames therefore does not replace continuing the computation when a cue must be recalled.

F Frozen-PaliGemma baselines on LIBERO

The LIBERO studies of Sections 4.4 and 4.6 keep PaliGemma frozen. This section provides separate reference configurations for that setting: two standard Transformer experts and LycoReco (reset) on frozen PaliGemma features.

The single-pass Transformer applies eight separately weighted decoder blocks once per observation; the recurrent-depth Transformer applies one decoder block eight times. Both start from eight memory tokens and 20 action queries of width 512. Each block has eight-head self-attention, cross-attention to the observation and a width-2048 feed-forward layer, and a linear map reads each query's action. LycoReco (reset) runs the CTM of Table 3 for $T = 8$ ticks. All three read the final-layer PaliGemma features through a trainable projection to width 512, together with a proprioception token (LycoReco uses this projection here instead of reading the VLM's keys and values), and restart at every observation. They predict a 20-step chunk with the L1 objective of Appendix B and train on batches of 32 observations, each from a different episode: 25k updates for the Transformers and 50k for LycoReco.

Each expert runs all fifty reference initial states of every LIBERO task (2,000 episodes), with the same scenes and seeds for every expert, and observes and acts at every step ($K = 1$).

Table 11: **Frozen-PaliGemma baselines on LIBERO.** Success (%) over 2,000 episodes per expert at $K = 1$. All experts reset at every observation; configurations differ from the budget sweep in Figure 4a. Params: trainable expert parameters, including input adapters; GFLOPs: expert computation per observation.

Expert	Params (M)	GFLOPs	Spatial	Object	Goal	Long	Mean
Single-pass Transformer	34.7	8.67	80.4	60.6	68.0	20.2	57.3
Recurrent-depth Transformer	5.3	8.67	75.2	63.8	56.6	19.0	53.7
LycoReco (reset)	3.9	2.07	77.0	65.4	63.0	22.6	57.0

The two Transformer experts reach 57.3% and 53.7% mean success on frozen PaliGemma, and LycoReco (reset) reaches 57.0%. This reset configuration uses projected features, $T = 8$, and 50k updates on batches of 32 observations from different episodes; the budget sweep reads final-layer keys and values directly and uses 20k updates on chronological windows containing 256 observations per update (Appendix D.5). Evaluation also uses 2,000 rather than 400 episodes, so this result is not a matched comparison with the sweep's one-tick carry result.

G Choice of action objective

We train every expert with L1 because it fits the tick recurrence cleanly: the loss supervises the pass of T ticks that produces the chunk, and the state after that pass is the state that carry keeps. We also consider two stochastic objectives, flow matching and the energy score.

L1. L1 penalizes the absolute error of each coordinate of the predicted chunk. In theory, a compromise between equally good chunks also minimizes it: for equally probable scalar targets $a^- < a^+$,

$$\tfrac{1}{2}|u - a^-| + \tfrac{1}{2}|u - a^+| = \tfrac{1}{2}(a^+ - a^-), \qquad u \in [a^-, a^+], \tag{6}$$

so every value between the two attains the minimum, and a chunk built from such values need not be a valid trajectory. Representing both chunks requires a distribution over chunks, which the two stochastic objectives provide.

Flow matching. Flow matching (Lipman et al., 2023) trains a velocity field. Each training example is one pass of ticks on a demonstration chunk noised to a random level, whereas at inference a chunk is integrated from noise with one pass per step, so N steps cost N passes of T ticks. The supervised pass is thus never the computation that produces a chunk, and carrying the state left after the last step would give the expert a state that training never produces.

Energy score. A stochastic expert maps the observation context and incoming state c, together with a noise draw $\xi \sim \mathcal{N}(0, I)$, to a chunk $A = g_\theta(c, \xi)$. With independent draws A, A' and demonstration A^*, the energy score (Gneiting & Raftery, 2007) is the loss

$$\mathcal{L}_{\mathrm{ES}} = \mathbb{E}\,\|A - A^*\|_2 - \tfrac{1}{2}\,\mathbb{E}\,\|A - A'\|_2. \tag{7}$$

The first term pulls draws toward the demonstration and the second keeps them apart, so the expected loss is minimized only when the draws follow the distribution of demonstrated chunks; a

compromise no longer minimizes it. One draw yields one chunk in a single pass of ticks, so, as with L1, the loss supervises the pass that produces the chunk.