IN-CONTEXT TEST TIME TRAINING FOR ADAPTIVE ROBOT POLICY LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

During deployment, robots continually receive new observations that can inform subsequent actions. Yet many robot policies operate with fixed parameters and a short observation history, limiting their use of accumulated experience. Learning from this experience requires both selecting useful information and training the policy to act with a memory that changes over time. We introduce In-Context Test Time Training (In-Context TTT), a framework that combines adaptive fast-weight memory with selective writing for robot policy learning. TTT layers encode incoming observations into fast weights, while a learned gate controls the strength of each memory update. To train this recurrent adaptation process, we develop Trajectory-Consistent Adaptation (TCA), which supervises action prediction along the fast-weight trajectories generated by ordered demonstrations. After chunk-level initialization, TCA carries fast weights across an entire episode, accumulates supervised gradients at successive adaptation states, and updates the slow parameters once at the episode boundary. This aligns the memory evolution used in training with deployment, where fast weights change while slow parameters remain fixed. On the six longest RoboCasa GR-1 tasks, selected by an average demonstration length of at least 500 steps, In-Context TTT achieves 53.3% mean success, compared with 50.0% for No-update SFT and 47.0% for the original policy. Real-robot evaluations obtain 7/10 successes in banana picking, 6/10 in cup stacking, and 4/10 in arranging digit-shaped blocks. Together, selective writing and trajectory-consistent supervision provide a way to learn from interaction history using demonstration data. Our core implementation is available at `https://anonymous.4open.science/r/In-Context-TTT-E538`.

1 INTRODUCTION

Robots continually receive new observations during deployment. A grasp reveals whether an object is secure, a placement changes the scene, and an unsuccessful attempt provides evidence for the next action. These observations can help a policy track progress and adjust its behavior throughout a task. Robot foundation models learn broad visuomotor knowledge from large datasets (Brohan et al., 2023; Octo Model Team et al., 2024; Kim et al., 2024; Black et al., 2024), but many are deployed with fixed parameters and a short observation history. Information outside that history cannot directly inform subsequent decisions. This raises a basic question: *how should a robot policy write and use the interaction context that accumulates during an episode?*

Effective use of context requires more than retaining observations. Consecutive frames often contain similar information, and an occluded or misleading view can interfere with previously useful evidence. Meanwhile, an unsuccessful action can reveal information that changes what the robot should do next. The value of an observation therefore depends on the task and the current memory. A robot policy needs a mechanism for adapting this memory and for controlling how strongly new observations change it.

Memory systems for language agents select information through extraction, consolidation, and retrieval (Chhikara et al., 2025). Fast-weight models instead encode context in parameters updated from the input (Schlag et al., 2021; Sun et al., 2024). RoboTTT demonstrates the value of this approach for long-context robot control (Jiang et al., 2026). We bring observation-dependent write selection into fast-weight adaptation: a learned gate modulates the proposed change to memory be-

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.

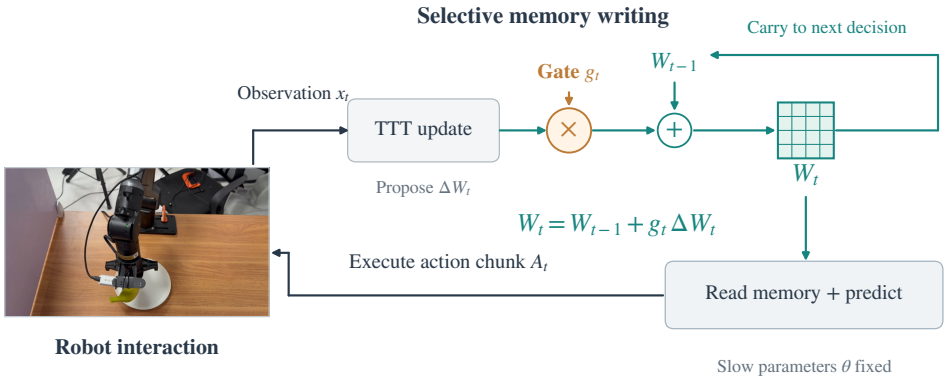


Figure 1: **In-Context TTT.** A banana-picking frame illustrates robot interaction: an observation-conditioned writer proposes a fast-weight update, and a learned gate controls its strength. The policy reads the updated memory to predict an action chunk. Fast weights carry interaction history across decisions and reset between episodes. The slow parameters remain fixed throughout deployment.

fore that change affects subsequent action generation. This provides a continuous, trainable way to control the influence of incoming information.

A second challenge is training the policy at the memory states it will use. Independent-chunk training with fast-weight updates disabled exposes the policy only to its initial memory. Deployment instead produces a sequence of states through repeated updates. The policy therefore needs supervision along this adaptation trajectory.

We introduce In-Context TTT, a framework that combines selective fast-weight memory with *Trajectory-Consistent Adaptation* (TCA). TCA treats the adaptation trajectory as the unit of supervised training. It processes a demonstration in temporal order, updates fast weights from each new observation, and supervises action prediction at the resulting state. The slow parameters stay fixed while the episode is processed. Gradients from its successive chunks accumulate and are applied once at the episode boundary, after which memory resets. This schedule preserves a single adaptation rule throughout each training trajectory, matching the separation between fixed slow parameters and evolving fast weights at deployment.

The training recipe first integrates TTT layers through chunk-level No-update SFT, then uses TCA to train repeated adaptation. Expert actions provide a learning signal at each step of the evolving memory trajectory, without requiring an online reward signal. The contribution of TCA is the combination of ordered memory evolution, supervision at the resulting adaptation states, and an episode-consistent slow-parameter schedule. Gradient accumulation implements that schedule.

On the six RoboCasa GR-1 tasks whose demonstrations average at least 500 steps, In-Context TTT reaches 53.3% mean success, improving on No-update SFT by 3.3 percentage points and the original policy by 6.3 points. The selected tasks span object placement into microwaves, cabinets, and tiered shelves. We further evaluate the final policy on physical banana picking, cup stacking, and ordered digit arrangement. Our contributions are:

- We formulate within-episode robot adaptation as learning to write and use a trajectory of fast-weight states, connecting interaction memory to history-conditioned action prediction.
- We introduce In-Context TTT, which combines observation-driven fast-weight updates with a learned write gate to control the influence of new observations on robot memory.
- We develop Trajectory-Consistent Adaptation, a supervised training procedure that learns action prediction along evolving fast-weight trajectories while keeping the slow parameters fixed within each episode.

2 PROBLEM FORMULATION

Let t index policy decisions and $x_t = (o_t, p_t)$ contain the visual observation and proprioception. Given instruction c, the policy predicts an action chunk $A_t = (a_{t,1}, \ldots, a_{t,H})$, executes its actions, and receives the next observation. The interaction history is $\mathcal{H}_t = (x_1, A_1, \ldots, A_{t-1}, x_t)$. We represent this history through a fast-weight state W_t:

$$W_0 = W_{\text{init}}, \qquad W_t = U_\theta(W_{t-1}, x_t, c), \qquad A_t \sim \pi_\theta(\cdot \mid x_t, c, W_t). \tag{1}$$

The current observation first updates memory, and the policy then reads the updated state to generate actions. The slow parameters θ govern the writer, gate, readout, and action model. They are fixed during deployment, while W_t evolves from observations and resets at episode boundaries. The memory size is fixed for a given architecture.

Training uses demonstrations $\mathcal{E} = \{(x_t, c, A_t^\star)\}_{t=1}^T$ drawn from dataset $\mathcal{D}$. Expert action chunks $A_t^\star$ supervise the action model; the inner memory update uses observation-derived features. This separates the information available to the writer at deployment from the action labels used to train the policy. We denote the supervised action loss by $\ell_{\text{action}}(\theta; x_t, c, A_t^\star, \zeta_t, W_t)$, where ζ_t contains the noise and timestep variables of the action-generation objective.

The adaptation state is part of the learning problem. Two decisions can have similar current observations but different histories and therefore different useful memory states. Time enters through the recurrent updates in Equation 1, rather than through an explicit timestamp. Independent-chunk training with fixed memory learns predictions at W_{init}. Recurrent deployment instead evaluates the policy at $W_1, W_2, \ldots, W_T$, each produced by its own update rule. Our training objective supervises the policy at these evolving states along recorded demonstration trajectories.

3 IN-CONTEXT TEST TIME TRAINING

3.1 ADAPTIVE MEMORY WITH SELECTIVE WRITING

We build In-Context TTT on the pretrained Kairos world-action model (Kairos Team, 2026), which couples a Video DiT with an Action DiT for world and action prediction. We insert TTT layers at selected self-attention-output positions of the Action DiT so that memory can directly influence action generation. The Action DiT and inserted TTT layers are trainable; the Video DiT, text encoder, and VAE remain frozen throughout our training.

Observation-driven adaptation. Let h_t denote the observation-conditioned features supplied to a TTT layer. Learned projections construct keys K_t, values V_t, and readout queries Q_t. A fast model f_W maps between the projected representations. Its reconstruction objective and proposed update are

$$\mathcal{L}_{\text{inner}}(W; h_t) = \frac{1}{N_t} \| f_W(K_t) - V_t \|_F^2, \qquad \widetilde{W}_t = W_{t-1} - \eta_\theta(h_t) \nabla_W \mathcal{L}_{\text{inner}}(W_{t-1}; h_t), \tag{2}$$

where N_t normalizes the reconstruction loss and η_θ controls the update scale. Both sides of the reconstruction objective come from the input. Thus, the fast-weight update requires neither expert actions nor task rewards during deployment. The supervised outer objective trains the surrounding representations and action model to use the resulting state for control.

Selective writing. A learned gate $g_t = G_\theta(h_t)$ controls how much of the proposed update is committed:

$$W_t = W_{t-1} + g_t \odot (\widetilde{W}_t - W_{t-1}), \qquad 0 \le g_t \le 1. \tag{3}$$

The multiplication broadcasts over the fast-weight dimensions controlled by the gate. A small gate preserves the existing state; a large gate incorporates more of the new information. This gives the policy a mechanism for limiting interference from an observation while retaining previously accumulated context. Selection acts on the memory update itself, so its effect persists into later decisions.

The memory readout is $h_t^{\mathrm{TTT}} = f_{W_t}(Q_t)$. It conditions action generation together with the current observation and instruction. The writer determines how the state changes, while the readout determines how that state affects the current action. In particular, reducing a write can preserve useful information for later chunks even when the current observation remains available through the policy's ordinary input path. This differs from an output gate, which directly scales the current memory contribution to the network output.

3.2 Chunk-Level Initialization

We initialize the adaptive policy through No-update SFT. Independent action chunks train the slow parameters with fast-weight updates disabled and no memory carried between samples:

$$\mathcal{L}_{\mathrm{init}}(\theta) = \mathbb{E}_{(x,c,A^\star)\sim\mathcal{D},\zeta}\left[\ell_{\mathrm{action}}(\theta; x, c, A^\star, \zeta, W_{\mathrm{init}})\right]. \tag{4}$$

This stage integrates the inserted TTT layers with the pretrained action model. "No-update" refers to observation-driven changes to fast weights; the slow parameters are optimized normally. The pretrained non-action components remain frozen, and an interpolation between attention and TTT outputs introduces the new layers during initialization. Implementation details appear in Appendix A.

This provides a policy that can generate actions through the TTT architecture. TCA then enables the fast-weight recurrence and trains action prediction along the resulting adaptation trajectories.

3.3 Trajectory-Consistent Adaptation

Learning along adaptation trajectories. For a demonstration episode $\mathcal{E}$, the current parameters θ induce a sequence of fast-weight states through Equation 1. We write this trajectory as

$$\mathcal{T}_\theta(\mathcal{E}) = \{(x_t, c, W_t^\theta, A_t^\star)\}_{t=1}^T, \qquad W_t^\theta = U_\theta(W_{t-1}^\theta, x_t, c). \tag{5}$$

TCA supervises actions along $\mathcal{T}_\theta(\mathcal{E})$, replacing training at a single initial state with training across the states reached by repeated adaptation. The expert supplies the observation sequence and action targets; the model's own writer generates the internal state trajectory. The state is therefore recomputed as the model changes across training episodes.

The trajectory objective is

$$\mathcal{L}_{\mathrm{TCA}}(\theta) = \mathbb{E}_{\mathcal{E}\sim\mathcal{D},\zeta_{1:T}}\left[\sum_{t=1}^{T} \ell_{\mathrm{action}}(\theta; x_t, c, A_t^\star, \zeta_t, W_t^\theta)\right]. \tag{6}$$

Every chunk contributes an action target at the adaptation state reached at that point in the episode. Later predictions are trained with the effects of earlier memory writes already present. The training distribution over fast states is induced jointly by the ordered demonstrations and the current update rule, rather than being fixed to the initialization.

One adaptation rule per episode. Let θ_e denote the slow parameters at the start of episode e. TCA holds them fixed throughout the episode and applies the following operations in order:

$$W_t = U_{\theta_e}(W_{t-1}, x_t, c), \tag{7}$$

$$\ell_t = \ell_{\mathrm{action}}(\theta_e; x_t, c, A_t^\star, \zeta_t, W_t), \tag{8}$$

$$G_t = G_{t-1} + \nabla_{\theta_e}\ell_t, \qquad G_0 = 0. \tag{9}$$

After the final chunk, the accumulated gradient updates the slow parameters:

$$\theta_{e+1} = \mathrm{OptimizerStep}(\theta_e, G_T), \qquad W \leftarrow W_{\mathrm{init}}. \tag{10}$$

Thus, fast weights evolve at the chunk timescale, while slow parameters evolve at the episode timescale. The procedure is summarized in Table 1.

Holding θ_e fixed makes the entire fast-weight trajectory consistent with one writer and one readout. If the slow parameters changed after every chunk, a state produced by one parameter setting would be passed to another, and a single training episode would mix successive adaptation rules. TCA keeps the rule fixed while exposing it to the changing context within an episode. The accumulated action losses then provide the supervised signal for the next slow update.

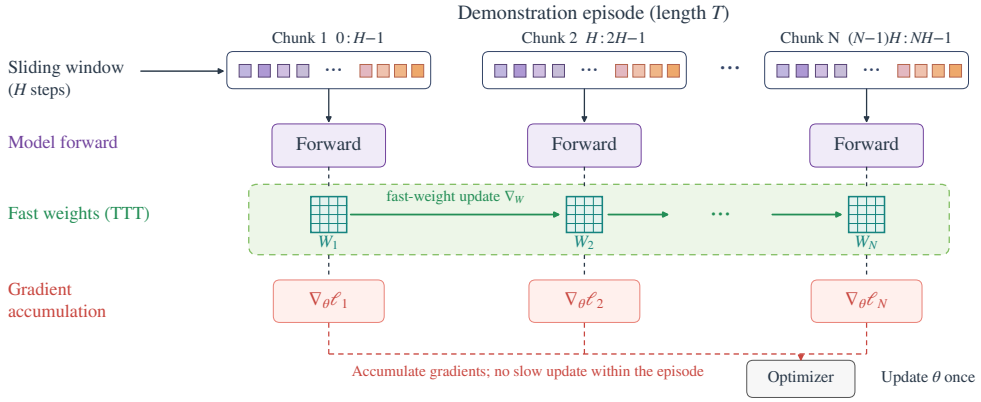


Figure 2: **Trajectory-Consistent Adaptation.** A demonstration episode is read in ordered chunks. Fast weights update and carry forward after each chunk, while each chunk contributes an action-loss gradient for the slow parameters. The gradients accumulate without changing the slow parameters within the episode. The optimizer applies one slow-parameter update after the final chunk; fast weights then reset.

Table 1: **TCA: training on one demonstration episode.**

Load episode $\mathcal{E}$; initialize $W = W_{\text{init}}$ and clear slow gradients.
For each chunk $t = 1, \ldots, T$, in temporal order:
1. Update W from the current observation using the learned writer and gate.
2. Predict actions with the updated W and compute supervised loss ℓ_t.
3. Accumulate the slow-parameter gradient; retain W for the next chunk.
After the final chunk: apply one slow optimizer step and reset fast weights.

Why trajectory consistency matters. The procedure addresses three aspects of memory use. First, temporal order determines which information is available at each decision. Second, persistent fast weights expose the action model to the states produced by repeated writes. Third, a fixed slow model ensures that these states are generated under the same parameter setting throughout the episode. These properties match the structure of deployment, where the policy continuously updates memory without taking slow optimizer steps.

Supervision over model-induced adaptation states. For each demonstration, the empirical distribution of training states is

$$d_\theta^{\text{adapt}}(\cdot \mid \mathcal{E}) = \frac{1}{T} \sum_{t=1}^{T} \delta_{(x_t, W_t^\theta, A_t^\star)}(\cdot), \tag{11}$$

where δ denotes a point mass. TCA supervises the policy over this model-induced distribution, including later states reached after repeated memory updates. In contrast, No-update SFT concentrates the memory component at W_{init}. As the writer changes across training episodes, TCA recomputes the adaptation trajectory and the states used for supervision. Expert action targets remain available throughout, providing a dense learning signal for using an evolving internal state. The alignment concerns memory dynamics along a trajectory; supervised observations come from demonstrations, while deployment observations arise from the robot's actions.

3.4 Deployment with Persistent Memory

Deployment follows the same observation-update-read sequence as TCA. Each environment owns a fast state initialized at the episode boundary. A new observation produces one committed memory update, after which the policy uses that state to generate the next action chunk. The state persists until the next decision and resets when the episode ends. Slow parameters remain fixed throughout execution.

Diffusion denoising requires repeated evaluations of the action model for a single observation. These evaluations share the committed fast state rather than treating repeated denoising calls as new observations. This keeps the memory trajectory tied to the interaction sequence. Separate state buffers likewise keep concurrent environments independent. The policy carries a fixed-size fast-weight state instead of storing the full sequence of past observations.

4 EXPERIMENTS

Our main experiments examine trajectory-consistent training and selective writing on Kairos, followed by three physical manipulation tasks. A preliminary study with a VLANeXt-based prototype complements these results by testing fast-weight adaptation under environment perturbations.

4.1 TASKS AND EVALUATION PROTOCOL

Long-horizon task selection. We first measure the mean demonstration length of all 24 RoboCasa GR-1 transfer tasks (Nasiriany et al., 2024). We evaluate every task with a mean length of at least 500 environment steps, yielding six tasks that span three source-object groups and three destination types. This criterion is fixed before comparing policies and selects tasks in which useful interaction evidence must be retained over many decisions. Appendix B reports the length of every candidate task and marks the selected subset.

Compared policies. The baseline is the original policy. No-update SFT integrates the TTT layers through independent-chunk supervision with fast-weight updates disabled during training. TCA starts from this initialization and trains with observation-driven writes and state carry across complete demonstration episodes. We compare the successive supervised stages using matched demonstration exposure, rather than equating an episode-level optimizer step with a chunk-level step.

Execution and metrics. The action model uses 32-step chunks, one 256×256 RGB frame, and ten denoising steps. Episodes are capped at 704 environment steps. We run 50 evaluation episodes per task and report the unweighted mean over the six tasks. The adaptive policy resets memory between episodes and retains it across consecutive action chunks.

4.2 MAIN RESULTS

Table 2: **Success rates (%) on six long-horizon RoboCasa GR-1 tasks.** The subset contains all tasks whose mean demonstration length is at least 500 steps. TCA is initialized from No-update SFT.

Task	Mean length	Baseline	No-update	TCA
Wine → CabinetClose	596.4	48.0	54.0	62.0
Potato → MicrowaveClose	576.4	50.0	52.0	52.0
Tray → Tieredshelf	528.6	38.0	30.0	40.0
Placemat → Tieredshelf	516.4	38.0	36.0	38.0
Milk → MicrowaveClose	513.4	52.0	64.0	56.0
Bottle → CabinetClose	506.6	56.0	64.0	72.0
Mean	539.6	47.0	50.0	**53.3**

TCA reaches 53.3% mean success, improving over No-update SFT by 3.3 percentage points and the original policy by 6.3 points. Relative to the baseline, TCA improves five tasks and matches one. The largest gains occur on Bottle → CabinetClose (+16 points) and Wine → CabinetClose (+14 points). These results show that the gain is not confined to a single source object or destination.

No-update SFT improves the mean from 47.0% to 50.0%, but its task-level effect is uneven. TCA further improves four tasks relative to No-update SFT, matches one, and decreases on Milk → MicrowaveClose. The progression therefore supports training with episode-persistent memory on average, while also showing that adaptation quality remains task dependent.

4.3 Selective Writing and Memory Use

We intervene on the write gate of the trained TCA policy at evaluation time. The normal condition uses the learned observation-dependent gate. The two controls override it with $g_t = 0$, which disables new fast-weight writes, or $g_t = 1$, which always commits the full proposed update. The three conditions form a separate matched evaluation from Table 2; within this comparison, all other parameters and evaluation settings remain unchanged.

Table 3: **Write-gate intervention on the six long-horizon tasks.** Values are success rates (%). "Learned" uses the observation-dependent gate; the fixed controls override it only during evaluation.

Task	Learned	$g_t = 0$	$g_t = 1$
Wine → CabinetClose	62	64	52
Potato → MicrowaveClose	52	48	48
Tray → Tieredshelf	40	40	40
Placemat → Tieredshelf	30	38	36
Milk → MicrowaveClose	56	58	58
Bottle → CabinetClose	72	60	68
Mean	**52.0**	51.3	50.3

The learned gate gives the highest average success, exceeding disabled writes by 0.7 points and unconditional writes by 1.7 points. Its effect varies by task: learned selection is especially helpful on Bottle and Potato, whereas fixed gates perform better on Placemat and Milk. The result supports controlling memory writes, but also indicates that the current gate does not uniformly identify the best update strength for every task.

4.4 Real-Robot Manipulation

We evaluate the final In-Context TTT policy on three physical tasks: picking a banana, stacking cups, and arranging digit-shaped blocks. The last task requires moving blocks shaped as 1, 2, and 3 and placing them in the order 1–2–3. For each task, we collect 200 demonstration trajectories and run 10 evaluation trials. Fast weights reset between trials and persist across action chunks within a trial.

Table 4: Real-robot data and evaluation results for the final In-Context TTT policy.

Task	Demonstrations	Trials	Success
Pick banana	200	10	7/10 (70%)
Stack cups	200	10	6/10 (60%)
Arrange digits 1–2–3	200	10	4/10 (40%)

Table 4 reports success in 7/10 banana-picking trials, 6/10 cup-stacking trials, and 4/10 digit-arrangement trials. These tasks test object acquisition, stacking, and ordered multi-object placement on a physical robot.

4.5 Preliminary Generalization Study

Before the Kairos experiments, we tested TTT in a world/action prototype built on VLANeXt (Wu et al., 2026). We compare attention with TTT mixers trained on clean demonstrations from all four LIBERO suites (Liu et al., 2023), then evaluate with LIBERO-Plus environment generation (Fei et al., 2025) under camera, robot-initial-state, lighting, background, and sensor-noise perturbations. Each fast-weight update uses 256 tokens, selected by the Spatial-only comparison in Appendix D. We report results first by suite and then by perturbation type.

TTT improves the five-perturbation average in every suite (Table 5), with the largest gains on Long (+16.9 points) and Goal (+14.4 points). The gains also span all five perturbation types (Table 6): background changes show the largest pooled improvement (+16.2 points), followed by lighting changes (+13.1 points), while camera changes still improve by 7.2 points. Because the number

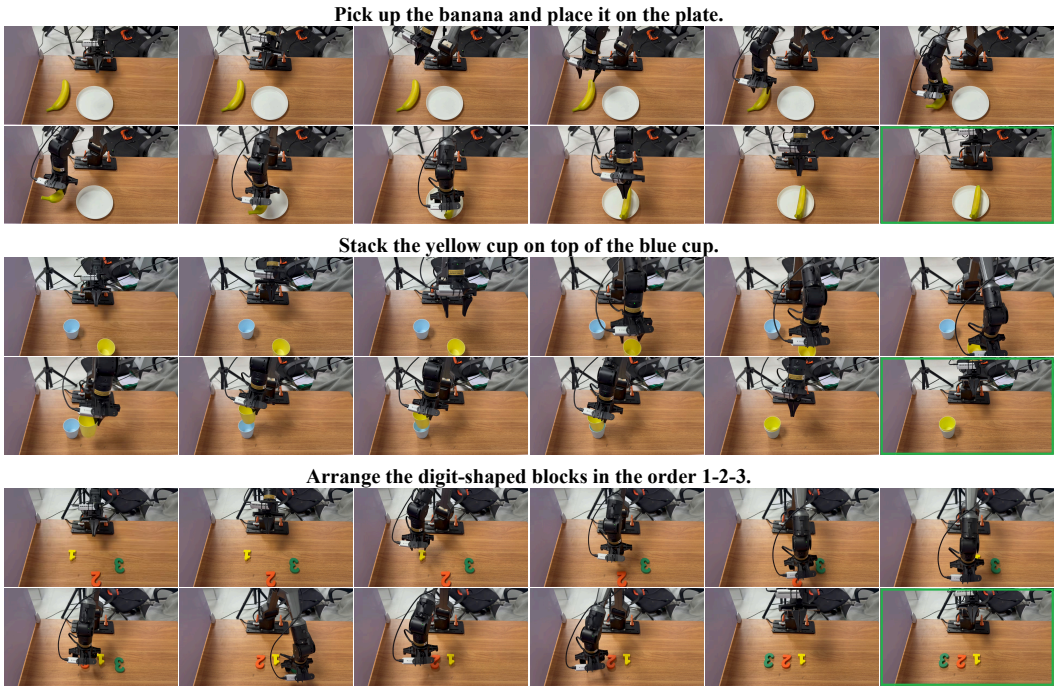


Figure 3: **Real-robot execution sequences.** Each task is shown in 12 frames, read from left to right across the first row and then the second. From top to bottom: banana picking, cup stacking, and arranging digit-shaped blocks in the order 1–2–3. Green borders mark the final frames.

Table 5: **Preliminary generalization by suite.** Success (%) on the VLANeXt-based prototype under LIBERO-Plus perturbations. Each suite averages its five perturbation types equally; the final row averages the four suites. Gains are percentage points.

Suite	Attention	TTT-256 (Ours)	Gain
Spatial	65.9	**72.6**	+6.7
Object	69.4	**74.6**	+5.2
Goal	54.7	**69.1**	+14.4
Long	39.9	**56.8**	+16.9
Mean	57.5	**68.3**	+10.8

of evaluated trials varies by perturbation, averaging the five pooled perturbation rates gives 68.1% versus 57.2%, whereas pooling all trials gives 65.8% versus 55.5%. The suite-level macro average is 68.3% versus 57.5% because it assigns equal weight to each suite before averaging. These preliminary results show that fast-weight adaptation can improve robustness to several forms of visual and initial-state variation, motivating our subsequent study of selective writing and trajectory-consistent training on Kairos.

5 Related Work

Robot policies and interaction context. Robot foundation models learn from diverse demonstrations (Brohan et al., 2023; Octo Model Team et al., 2024; Kim et al., 2024; Liu et al., 2025; Black et al., 2024), while diffusion policies generate continuous actions through iterative denoising (Chi et al., 2023). We study how pretrained action models use deployment history through recurrent fast weights, complementing advances in architecture and pretraining.

Test-time training for sequence modeling and control. Test-time training adapts models using input-derived supervision (Sun et al., 2020); TTT sequence layers implement recurrent fast-weight

Table 6: **Preliminary generalization by perturbation.** Pooled success (%) across four LIBERO suites. Parentheses give the number of trials per method; gains are percentage points. Macro averages five perturbations; pooled weights all 6,968 trials.

Perturbation	Attention	TTT-256 (Ours)	Gain
Camera (1599)	32.8	40.0	+7.2
Robot initial state (1550)	50.4	59.9	+9.5
Lighting (1142)	73.6	86.7	+13.1
Background (1076)	67.3	83.5	+16.2
Sensor noise (1601)	62.2	70.5	+8.3
Macro average (6968)	57.2	68.1	+10.9
Pooled trials (6968)	55.5	65.8	+10.3

updates (Schlag et al., 2021; Sun et al., 2024; Zhang et al., 2025). RoboTTT scales robot context through sequence action forcing and truncated backpropagation (Jiang et al., 2026). Its integration gate scales the TTT output, whereas our gate controls the memory update. TCA trains action prediction along ordered fast-weight trajectories with one slow-parameter update per episode.

Memory selection and continual adaptation. Agent memory systems select retained information (Chhikara et al., 2025), and continual-learning methods address interference between updates (Aljundi et al., 2019). Our fast state retains within-episode context while the slow model preserves knowledge across episodes. Selective writing and TCA connect this memory to supervised action learning.

6 Discussion

Training on the states used at deployment. TCA improves No-update SFT by 3.3 points. Training on evolving memory states while holding slow parameters fixed within an episode gives the writer and readout a consistent adaptation rule.

Interpreting selective writing. The learned gate exceeds disabled writes by 0.7 points and unconditional writes by 1.7 points on average, with varying effects across tasks. This motivates learning both useful memory updates and when to apply them.

7 Conclusion

We introduced In-Context TTT with selective memory writing and Trajectory-Consistent Adaptation. By supervising actions along evolving fast-weight states, TCA raises mean success on six long-horizon RoboCasa tasks from 50.0% after initialization to 53.3%, compared with 47.0% for the original policy. Gate interventions and three physical manipulation tasks further evaluate the resulting adaptive policy.

Reproducibility Statement

Section 3 describes the method; the appendix gives implementation and evaluation details. Our core implementation is available at `https://anonymous.4open.science/r/In-Context-TTT-E538`.

AI Use Disclosure

We used generative AI to assist with experimental code, figures, and manuscript editing. AI did not originate the research ideas; the authors reviewed all AI-assisted material and take responsibility for this work.

REFERENCES

Rahaf Aljundi, Eugene Belilovsky, Tinne Tuytelaars, Laurent Charlin, Massimo Caccia, Min Lin, and Lucas Page-Caccia. Online continual learning with maximal interfered retrieval. *Advances in neural information processing systems*, 32, 2019.

Kevin Black, Noah Brown, Danny Driess, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Anthony Brohan, Noah Brown, Justice Carbajal, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, 2023.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Robotics: Science and Systems*, 2023.

Senyu Fei, Siyin Wang, Junhao Shi, Zihao Dai, Jikun Cai, Pengfang Qian, Li Ji, Xinzhe He, Shiduo Zhang, Zhaoye Fei, Jinlan Fu, Jingjing Gong, and Xipeng Qiu. LIBERO-Plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

Yunfan Jiang, Yevgen Chebotar, Ruijie Zheng, Fengyuan Hu, Yunhao Ge, Jimmy Wu, Tianyuan Dai, Scott Reed, Fei-Fei Li, Yuke Zhu, and Linxi Fan. RoboTTT: Context scaling for robot policies. *arXiv preprint arXiv:2607.15275*, 2026.

Kairos Team. Kairos: A regret-aware native world-action model stack for physical AI. *arXiv preprint arXiv:2606.16533*, 2026.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, et al. OpenVLA: An open-source vision-language-action model. In *Conference on Robot Learning*, 2024.

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

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. RDT-1B: A diffusion foundation model for bimanual manipulation. In *International Conference on Learning Representations*, 2025.

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

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, et al. Octo: An open-source generalist robot policy. In *Robotics: Science and Systems*, 2024.

Imanol Schlag, Kazuki Irie, and Juergen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, 2021.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International Conference on Machine Learning*, 2020.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

Xiao-Ming Wu, Bin Fan, Kang Liao, Jian-Jian Jiang, Runze Yang, Yihang Luo, Zhonghua Wu, Wei-Shi Zheng, and Chen Change Loy. VLANeXt: Recipes for building strong VLA models. *arXiv preprint arXiv:2602.18532*, 2026.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T. Freeman, and Hao Tan. Test-time training done right. *arXiv preprint arXiv:2505.23884*, 2025.

A Implementation Details

Model components. We use Kairos (Kairos Team, 2026) as the pretrained world-action backbone. Its Video DiT and Action DiT model future visual and action tokens, respectively. We insert TTT layers after the self-attention outputs of Action-DiT blocks $[3, 7, 11, 15, 19, 23, 27]$ (zero-indexed). During our training, the Action DiT and the inserted TTT layers are optimized, while the Video DiT, text encoder, and VAE remain frozen.

Training and evaluation settings. Both stages use AdamW with a learning rate of 5×10^{-6}, zero weight decay, and $\epsilon = 10^{-12}$ on eight GPUs. No-update SFT runs for 20,000 optimizer steps with one action chunk per GPU, a 5,000-step warmup, and cosine decay to 10^{-6}. TCA runs for 9,000 optimizer steps with a 1,000-step warmup and one episode stream per GPU; a 576-frame stream contains up to 18 consecutive chunks. The fast-weight update uses a fixed base step size of 0.01. At each decision, the policy receives one 256×256 RGB frame and predicts and executes 32 actions using 10 denoising steps. We evaluate each task for 50 trials with a limit of 704 environment steps and report the unweighted mean success rate over the six selected tasks.

Initialization. No-update SFT uses independently sampled chunks with observation-driven fast-weight writes disabled. The pretrained attention output is progressively interpolated with the TTT output:

$$h_{\text{out}} = (1 - \alpha_s)h_{\text{attn}} + \alpha_s h_{\text{TTT}}. \tag{12}$$

The coefficient α_s follows the interpolation schedule used during the 20,000-step initialization stage. It controls the integration of TTT into the pretrained network. The write gate in Equation 3 instead acts on the change to fast memory in response to an observation.

Trajectory-consistent training. TCA processes each episode stream in temporal order. At each chunk, observation-derived features update the fast weights before action prediction and supervised gradient computation. The fast state is carried forward, detached, and cloned between chunks, giving a recurrent gradient horizon of one chunk. Slow parameters remain fixed throughout the stream. We sum the chunk gradients without dividing by the number of chunks or valid action tokens, clip the accumulated gradient to a maximum norm of 1.0, and take one optimizer step at the end of the stream. Fast weights are then reset for the next stream.

Execution. An environment retains its own fast state. One observation-driven update is committed per decision; the action model shares the resulting state across the denoising evaluations for that decision. Episode termination resets the state. This schedule is shared by supervised sequence processing and deployment.

B RoboCasa Task Lengths

We compute the mean demonstration length for each of the 24 RoboCasa GR-1 transfer tasks before evaluating the policy variants. The main evaluation includes every task with a mean length of at least 500 environment steps. Table 7 reports the complete set; bold rows satisfy the selection rule. The selected tasks cover six source–destination pairs rather than a single object family.

Table 7: Mean demonstration lengths for all 24 candidate tasks. Bold rows form the long-horizon evaluation set (≥ 500 steps).

Category	Task	Length
PnP	**Wine → CabinetClose**	**596.4**
	Potato → MicrowaveClose	**576.4**
	Milk → MicrowaveClose	**513.4**
	Bottle → CabinetClose	**506.6**
	Cup → DrawerClose	456.4
	Can → DrawerClose	379.6
Cuttingboard	Cuttingboard → Cardboardbox	495.0
	Cuttingboard → Tieredbasket	467.0
	Cuttingboard → Basket	457.2
	Cuttingboard → Pot	410.8
	Cuttingboard → Pan	301.0
Placemat	**Placemat → Tieredshelf**	**516.4**
	Placemat → Basket	426.4
	Placemat → Bowl	390.8
	Placemat → Plate	362.4
Plate	Plate → Pan	438.0
	Plate → Cardboardbox	437.4
	Plate → Bowl	365.0
	Plate → Plate	319.4
Tray	**Tray → Tieredshelf**	**528.6**
	Tray → Tieredbasket	404.2
	Tray → Cardboardbox	368.4
	Tray → Pot	353.0
	Tray → Plate	334.0

Within each group, tasks are sorted by decreasing mean length. The six selected tasks have a mean length of 539.6 steps.

C Write-Gate Intervention

Table 3 changes only the gate used by the trained TCA checkpoint during a separate matched evaluation. The learned condition uses $G_\theta(h_t)$. Setting $g_t = 0$ preserves the incoming state and suppresses the proposed write, whereas $g_t = 1$ commits the full proposed update. Fast-state persistence, action sampling, episode limits, and all model parameters are identical across the three gate conditions. This intervention measures whether the observation-dependent gate improves over two extreme, input-independent write policies without conflating the comparison with retraining.

D Preliminary Study on VLANeXt

D.1 Prototype and Evaluation

Our preliminary world/action prototype is built on VLANeXt (Wu et al., 2026). We replace the token mixer in every fourth Transformer block with a multi-head SwiGLU TTT mixer, giving seven TTT mixers in a 29-block network while retaining the projections, conditioning, and MLPs. The visual expert uses block-causal mixing, and the action expert jointly mixes action tokens and visual conditions during denoising. Fast weights evolve within a sequence computation. This prototype

tests adaptive token mixing; the Kairos experiments study the subsequent episode-persistent policy and TCA training procedure.

D.2 Tokens per Fast-Weight Update

We first train on clean LIBERO-Spatial demonstrations and evaluate 1,627 perturbed task variants generated with LIBERO-Plus. Table 8 compares attention with TTT using 16 or 256 tokens per fast-weight update. This token count controls the amount of sequence information used in one inner update and is distinct from the number of robot actions executed per decision.

Table 8: **Selecting the token count per TTT update.** Success rates (%) on perturbed Spatial tasks using the VLANeXt-based prototype. TTT-16 and TTT-256 use 16 and 256 tokens per update, respectively.

Perturbation	Attention	TTT-16	TTT-256
Camera	70.2	69.1	**78.7**
Light	98.6	**99.0**	**99.0**
Background	**98.1**	96.5	97.7
Noise	80.9	**98.0**	95.4
Robot	69.4	80.3	**82.9**
Overall	81.9	87.5	**89.9**

TTT-256 achieves the highest overall success: 89.9%, compared with 87.5% for TTT-16 and 81.9% for attention. Its gains over attention are largest under robot-initial-state, noise, and camera perturbations; TTT-16 performs best under noise. We select 256 tokens per update for the subsequent mixed-suite study in Section 4.5. The same token-update size is used in our Kairos implementation.