# SELA-TTT: SEMANTICALLY MODULATED TEST-TIME TRAINING FOR VISION-LANGUAGE-ACTION MODELS

**Anonymous authors**
Paper under double-blind review

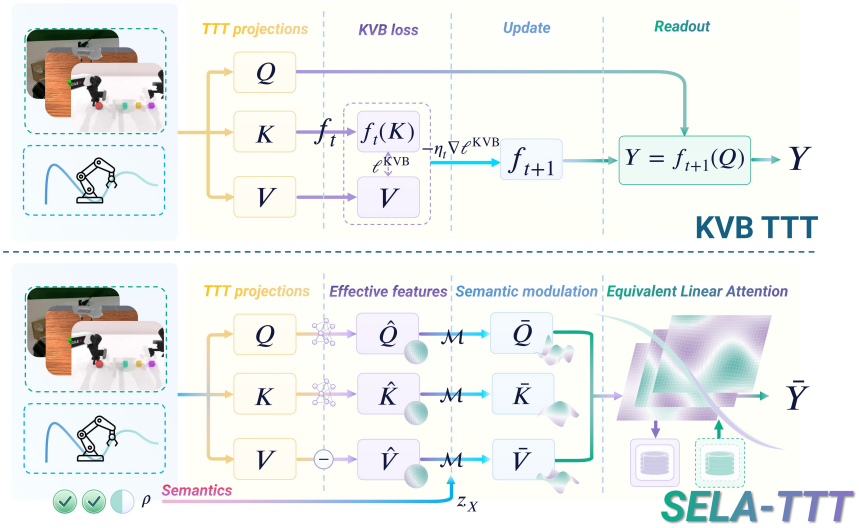



Figure 1: Semantic modulation of TTT. SELA-TTT uses explicit semantic signals to condition context writes and reads through TTT's equivalent linear-attention formulation.

## ABSTRACT

Context enables Vision-Language-Action (VLA) models to use past interactions and demonstrations during task execution. Test-Time Training (TTT) represents this information through recurrent fast-weight updates, but conventional KV-binding updates leave implicit how each interaction's semantics should guide its contribution to context. We introduce Semantically Modulated Equivalent Linear Attention for Test-Time Training (**SELA-TTT**), which uses explicit semantic signals to relate these contributions to task structure. Our method modulates the queries, keys, and values in TTT's equivalent, time-evolving linear-attention formulation, conditioning both context writes and reads. The resulting fast state carries semantic information forward to subsequent decisions. We instantiate this interface with subtask advancement, overall goal progress, and atomic-skill optimality, capturing complementary aspects of task progress and execution validity. Evaluations across simulation benchmarks and real-world task suites show improved success over TTT baselines, with equivalent linear-attention modulation outperforming injection into input features or raw TTT projections. Controlled label perturbations further support the lasting influence of semantics through their correspondence with interaction history. Beyond stronger task performance, **SELA-TTT** enables semantic-aware in-context learning from imperfect human demonstrations and generalization to previously unseen failures.

## 1 INTRODUCTION

Context allows Vision-Language-Action (VLA) models to use demonstrations and preceding interactions when generating subsequent actions. This information supports long-horizon execution and



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.

adaptation beyond the current observation (Wang et al., 2023; Jain et al., 2024; Jiang et al., 2026). Recent work has increasingly explored Test-Time Training (TTT) as a mechanism for representing such context (Sun et al., 2025; Feng et al., 2026; Jiang et al., 2026). By making the recurrent state an adaptive neural network, TTT compresses interaction history through self-supervised parameter updates. Its fast weights provide an expressive, fixed-size representation that evolves with the sequence, supporting both demonstration conditioning and online adaptation (Sun et al., 2025; Jiang et al., 2026).

However, an expressive representation of history does not by itself make the semantics of its contents explicit. In KV-binding TTT, learned projections produce the queries, keys, and values used in fast-model computation. The fast model is updated through a self-supervised objective defined on the projected keys and values, then read through the queries (Sun et al., 2025; Liu et al., 2026a). Although action-supervised learning can make these features useful for control (Jiang et al., 2026), how task progress and execution validity should guide each update remains implicit. To explicitly relate the updates and readouts of TTT-represented context to task structure, we believe that an effective interface for modulating projected features with semantic signals is essential.

Building upon this insight, we introduce Semantically Modulated Equivalent Linear Attention for Test-Time Training (**SELA-TTT**), which relates the contributions of interaction history to task structure by semantically modulating features in TTT's equivalent linear-attention computation. This formulation exposes the effective queries, keys, and values governing context writes and reads (Liu et al., 2026a). By modulating these representations, SELA-TTT directly conditions both fast-state updates and the readout used for action generation on task structure conveyed by semantic signals, bypassing semantic conditioning through input features or raw TTT features before the fast-model transformation.

With this modulation, SELA-TTT can retain interactions that advance a subtask, deviate from it, or correct an error while distinguishing their influence on subsequent actions. Explicit semantics provide a basis for preserving these distinctions as interactions accumulate, allowing context to guide execution according to task requirements (Liu et al., 2026b; Li et al., 2025). The modulated fast state is carried forward, so an interaction's semantics can affect later decisions rather than only the immediate response to feedback. To capture the varied task structure of long-horizon robotic manipulation, we instantiate this interface not only with current subtask advancement, but also with overall goal progress to extend its temporal scope and atomic-skill optimality to distinguish execution validity across action primitives.

We expect SELA-TTT to improve task performance by aligning contextual updates and readouts with task progress and skill execution validity. Beyond these gains, making task structure explicit allows semantic modulation to situate even unseen interactions within that structure and guide subsequent actions accordingly. This provides a basis for handling failures involving unseen physical interactions or unseen semantic–interaction correspondences. The same mechanism extends to demonstration context: semantic annotations distinguish the roles of demonstrated interactions, enabling selective imitation from an imperfect video prompt.

Experiments across RoboMME (Dai et al., 2026), RMBench (Chen et al., 2026), and real-world task suites confirm these expected performance gains: semantic modulation improves success over TTT baselines trained with additional expert data or failure-to-correction trajectories. Comparisons across injection interfaces favor the equivalent linear-attention formulation, while read–write ablations show that jointly modulating reading queries and writing keys and values performs best. Perturbing prior semantic labels further shows that their influence persists through their correspondence with interaction history. Real-world evaluations further confirm the anticipated capabilities, demonstrating semantic-aware in-context learning from imperfect human demonstrations and generalization to previously unseen failures.

## 2 Related Work

**Long-Context Visuomotor Policy Learning.** Recent long-horizon visuomotor policies, including VLA models, represent context through historical observations, compressed features, or adaptive model states. EventVLA (Yang et al., 2026a) retains selected frames, while HALO (Shah et al., 2026) and Gated Memory Policy (Gao et al., 2026) encode observation–action histo-

ries. SAM2Act+ (Fang et al., 2025), MemoryVLA (Shi et al., 2025), and Chameleon (Guo et al., 2026) construct dedicated visual or event memories. ContextVLA (Jang et al., 2025), HAMLET (Koo et al., 2025), NativeMEM (Wang et al., 2026b), and VPWEM (Lei et al., 2026) further compress history into conditioning tokens. Rather than retaining individual entries, MTIL (Zhou et al., 2025), SeedPolicy (Gui et al., 2026), ReMem-VLA (Li et al., 2026), and $\mu$VLA (Cherepanov et al., 2026) integrate context through recurrent hidden states. WAM-TTT (Feng et al., 2026) and RoboTTT (Jiang et al., 2026) adopt TTT (Sun et al., 2025) for recurrent context compression through neural-network parameter updates, using human-video context and demonstrations with manipulation histories, respectively. These approaches develop representations of interaction history for action generation. We focus on explicitly relating the contributions of that history to task structure through semantic modulation of TTT.

**Test-Time Training for Context Modeling.** Recurrent context models accumulate information through increasingly expressive state updates. Linear attention (Katharopoulos et al., 2020) accumulates a matrix state, with its fast-weight interpretation (Schlag et al., 2021) connecting this computation to associative learning and delta-rule corrections. Gated DeltaNet (Yang et al., 2025) adds input-dependent state control, while Longhorn (Liu et al., 2025) and Test-Time Regression (Wang et al., 2025) formulate recurrence through online learning and regression. TTT (Sun et al., 2025) makes the state a learnable model whose parameters compress context through self-supervised updates, supporting performance gains with longer context. Titans (Behrouz et al., 2025b) and ATLAS (Behrouz et al., 2025a) develop neural memory, while LaCT (Zhang et al., 2025) and TTT-E2E (Tandon et al., 2025) advance scalable optimization and end-to-end training. RoboTTT (Jiang et al., 2026) and WAM-TTT (Feng et al., 2026) bring these representations to robot policies. Complementing these architectures, TTT-is-LA (Liu et al., 2026a) identifies the equivalent time-evolving linear-attention computation induced by KV-binding updates. We use this formulation as an interface for semantic modulation of context writes and reads.

**Semantic-Aware Robot Learning.** Semantic-aware learning incorporates semantic structure into representation, supervision, and action generation (JoyAI-RA Team, 2026). BPP (Mark et al., 2026), MemoryVLA (Shi et al., 2025), and MEM (Torne et al., 2026) introduce semantic information into context representations, while HALO (Shah et al., 2026), PALM (Liu et al., 2026b), and DiM-WAM (Wang et al., 2026a) add semantic supervision through question answering or progress prediction. Semantics also guide behavior through instruction generation in MemER (Sridhar et al., 2025), phase-dependent planning in Mem-0 (Chen et al., 2026), and history-aware verification in HAVE (Li et al., 2025). ProgressVLA (Yan et al., 2026), Long-VLA (Fan et al., 2025), and TFP (Liang et al., 2026) extend this guidance into action generation. These studies incorporate semantics through context representations, auxiliary supervision, planning, or action generation. Our focus is semantic modulation within TTT's recurrent computation, where task progress and execution validity guide the contributions written into the fast state and subsequently read for action generation.

## 3 Problem Formulation

**Robotic interaction.** We model robotic interaction as a POMDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, P_T, P_O, R, \mu_0, \gamma)$, where $\mathcal{S}$ comprises the underlying robot and environment states, $\mathcal{A}$ is the action space, and $\mathcal{O}$ contains visual and proprioceptive observations together with an episode-level language instruction $c$. The transition and observation models are $P_T$ and $P_O$, respectively; $R$ is the reward function, $\mu_0$ the initial-state distribution, and $\gamma$ the discount factor. Environment steps are indexed by $n$, while $t$ indexes decision boundaries at steps $n_t$.

$$s_{n+1} \sim P_T(\cdot \mid s_n, a_n), \qquad r_n = R(s_n, a_n), \qquad \rho_t = E_{\text{sem}}(s_{n_t}). \tag{1}$$

**Semantic-conditioned action generation.** Let $h_t = (o_n, a_n)_{0 \le n < n_t}$ denote the interaction history available at decision boundary $n_t$. The semantic signal $\rho_t = E_{\text{sem}}(s_{n_t})$ describes the preceding executed segment and is available before $A_t$ is generated. We seek a VLA model that uses this history and its associated semantic signals, together with the current observation, to generate action chunks,

$$A_t \sim \pi(\cdot \mid o_{n_t}, h_t, \rho_{1:t}), \qquad A_t = [a_{n_t}, \ldots, a_{n_t+H-1}] \in \mathcal{A}^H. \tag{2}$$

At each decision boundary, the model generates a chunk of $H$ actions and sequentially executes a fixed prefix.

**Semantic modulation of context.** TTT represents interaction history through recurrent fast-weight updates (Sun et al., 2025; Jiang et al., 2026). Our objective is to make task progress and execution validity explicit in how this history contributes to subsequent actions. This requires an interface through which semantic signals guide both the accumulation of the fast state and its readout. The following section develops such an interface through TTT's equivalent linear-attention formulation.

# 4 METHODOLOGY

## 4.1 TEST-TIME TRAINING AND ITS LINEAR-ATTENTION NATURE

We express KV-binding TTT in its equivalent linear-attention form to identify the representations governing context writes and reads. These representations provide the interface for semantic modulation introduced in Section 4.2.

At an action-chunk boundary $t$, learned embeddings of hidden tokens $X_t \in \mathbb{R}^{L\times d}$, with $L$ tokens of width $d$, produce queries $Q_t$, keys $K_t$, and values $V_t$. KV-binding TTT (Sun et al., 2025) binds each key to its value with a self-supervised objective, updating a fast model $f_t = f_{\theta_t}$ before reading it with the query,

$$\ell_t^{\mathrm{KVB}}(\theta_t) = \frac{1}{2}\|f_t(K_t) - V_t\|_F^2, \qquad \theta_{t+1} = \theta_t - \eta_t \nabla_{\theta_t} \ell_t^{\mathrm{KVB}}(\theta_t), \qquad Y_t = f_{t+1}(Q_t), \tag{3}$$

Here, $\theta_t$ denotes the fast-model parameters, $\eta_t$ the inner learning rate, and $Y_t$ the TTT output features for the tokens in $X_t$. Following RoboTTT's (Jiang et al., 2026) integration practice, we embed TTT layers into a pretrained $\pi_{0.5}$ (Physical Intelligence et al., 2025) VLA model by augmenting every action-expert block with a gated residual between joint attention and the feed-forward sublayer,

$$X_t' = X_t + \tanh(g) \odot P(Y_t). \tag{4}$$

Here $X_t$ contains the post-attention action and register features, $P$ is an output projection, and $g$ is a learned channel-wise gate, with the layer index omitted. Figure 2 summarizes this integration and the semantic modulation introduced in Section 4.2; Appendix C provides architecture and initialization details.

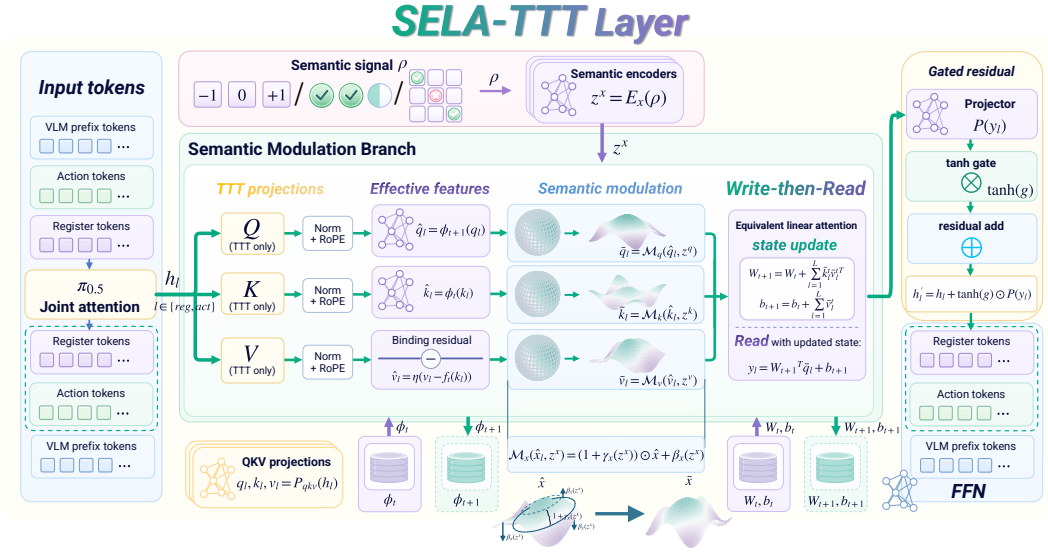



Figure 2: Overview of SELA-TTT. Semantic signals modulate effective queries, keys, and values for the fast-state update and readout, whose output returns through a gated residual.

Our diagnostics, adapted from TTT-is-LA (Liu et al., 2026a), challenge a literal raw-KV fitting interpretation when TTT is integrated into VLA layers (Appendix A). Following its linear-attention analysis, we express the fast model, with hidden representation $\phi_t(z)$ and final-layer weight matrix $W_t$ and bias $b_t$, as

$$f_t(z) = W_t^\top \phi_t(z) + b_t. \tag{5}$$

Unrolling the affine updates gives the following history-dependent, time-evolving linear-attention readout:

$$y_{t,\ell} = W_0^\top \widehat{\mathbf{q}}_{t,\ell} + b_0 + \sum_{i=0}^{t} \sum_{j=1}^{L} \widehat{\mathbf{v}}_{i,j} \big( \widehat{\mathbf{k}}_{i,j}^\top \widehat{\mathbf{q}}_{t,\ell} + 1 \big), \tag{6}$$

in which we identify representations that serve as its queries, keys, and values:

$$\widehat{\mathbf{q}}_{t,\ell} = \phi_{t+1}(q_{t,\ell}), \qquad \widehat{\mathbf{k}}_{t,j} = \phi_t(k_{t,j}), \qquad \widehat{\mathbf{v}}_{t,j} = -\eta_t \frac{\partial \ell_t^{\mathrm{KVB}}}{\partial f_t(k_{t,j})} = \eta_t \big( v_{t,j} - f_t(k_{t,j}) \big), \tag{7}$$

where $\ell$ and $j$ index current and context tokens, respectively; we omit the head index throughout. We call these the effective queries, keys, and values. The full derivation appears in Appendix B.1, with numerical and behavioral equivalence tests in Appendix B.2. Unlike raw input projections, these effective representations appear directly in the context update and readout (Liu et al., 2026a, Section 5.2). Modulating them therefore conditions how interactions contribute to the evolving fast state and subsequent action generation. Table 16 compares this interface with conditioning the input feature space or raw TTT space. We do not adopt the architectural simplifications proposed in that analysis (Liu et al., 2026a), as they reduce success in our setting (Appendix C.6).

### 4.2 Semantic Modulation of Equivalent Linear Attention

To relate contextual contributions to task structure, SELA-TTT uses explicit semantic signals to modulate the effective queries, keys, and values. Semantic information thereby guides both the fast-state updates and the readout used for subsequent action generation. At each TTT layer, SELA-TTT uses separate encoders $E_q, E_k, E_v$ to embed the available signal $\rho_t$ and modulate the corresponding effective features. Omitting the layer index,

$$z_t^x = E_x(\rho_t), \qquad \overline{x}_t = \mathcal{M}_x(\widehat{x}_t, z_t^x), \qquad x \in \{q, k, v\}. \tag{8}$$

**Semantic signals.** We consider three semantic signals that describe complementary aspects of task structure. Subtask advancement captures progress within the active subtask, overall goal progress tracks the requirements remaining across the task, and atomic-skill optimality associates execution validity with the identity of the active primitive. Comparing these signals examines which distinctions most effectively guide TTT's use of context.

*1) Subtask advancement.* Subtask advancement distinguishes progress toward the active subtask from deviation and corrective intervention. Let $\mathcal{P}_t$ and $\mathcal{D}_t$ denote advancement and deviation, respectively:

$$\rho_t^{\mathrm{adv}} = \mathbf{1}[\mathcal{P}_t] - \mathbf{1}[\mathcal{D}_t]. \tag{9}$$

Here, $\mathbf{1}[\cdot]$ is the indicator function; both terms are zero during corrective intervention. Appendix D.1.3 specifies the annotation procedure.

*2) Overall goal progress.* Overall goal progress tracks the requirements remaining across the task:

$$\rho_t^{\mathrm{goal}} = N - C_t, \tag{10}$$

where $N$ and $C_t$ count required and completed subtasks. The temporal separation inherent in the write–then–read computation allows this signal to describe task progress at distinct stages of the update; we condition writing on the previous goal state and reading on the current one,

$$z_t^q = E_q(\rho_t^{\mathrm{goal}}), \qquad z_t^k = E_k(\rho_{t-1}^{\mathrm{goal}}), \qquad z_t^v = E_v(\rho_{t-1}^{\mathrm{goal}}). \tag{11}$$

*3) Atomic skill optimality.* Atomic-skill optimality associates execution validity with the active primitive, distinguishing which skill is evaluated in a shared primitive space:

$$\rho_t^{\mathrm{skill}}[j] = \xi_t \, \mathbf{1}[j = j_t], \qquad \xi_t \in \{-1, +1\}, \quad j \in \{1, \ldots, J\}, \tag{12}$$

where $J$ counts atomic skills and $j_t$ is the active skill. Here, optimality denotes execution validity: $\xi_t = -1$ when the advancement label is negative and $+1$ otherwise (Appendix D.1.3). Inactive slots are zero, preserving primitive identity.

**Semantic Modulation.** Our aim is to modulate contextual contributions within the effective feature space that directly governs context writes and reads. We thus use FiLM (Perez et al., 2018) to generate

channel-wise scales and shifts, allowing semantics to adjust how an interaction is represented within this space (Gao et al., 2025; Zhuang et al., 2026). For $x \in \{q, k, v\}$,

$$\overline{x} = (1 + \gamma_x(z^x)) \odot \widehat{x} + \beta_x(z^x). \tag{13}$$

Figure 9 in Appendix B.3 illustrates the channel-wise transformation. With $\overline{v}$ denoting the post-mask value, masked tokens contribute no weight or bias update. This modulation allows interactions that advance a subtask, deviate from it, or correct an error to contribute differently to subsequent actions.

The resulting recurrence is (Appendix B.3)

$$W_{t+1} = W_t + \sum\nolimits_{j=1}^{L} \overline{k}_{t,j}\overline{v}_{t,j}^{\top}, \quad b_{t+1} = b_t + \sum\nolimits_{j=1}^{L} \overline{v}_{t,j}, \quad y_{t,\ell} = W_{t+1}^{\top}\overline{q}_{t,\ell} + b_{t+1}. \tag{14}$$

The parameters of $\phi_t$ retain their KVB updates, while semantics modify the affine write and readout. The updated fast state is carried into subsequent chunks, allowing the semantic meaning of an interaction to influence later decisions through the accumulated context.

# 5 Experiments

We evaluate whether explicit semantic modulation improves TTT-based action generation and how the choice of modulation interface affects its use of context. RoboMME compares three semantic signals and injection interfaces, with read–write ablations and prior-label perturbations examining how semantics affect the fast state and subsequent behavior. RMBench tests the approach under different memory demands. Real-world experiments assess task performance, semantic-aware in-context learning from imperfect demonstrations, and generalization to unseen failures.

## 5.1 Experiments on RoboMME

**Benchmark.** RoboMME (Dai et al., 2026) evaluates context-dependent manipulation in a ManiSkill-based tabletop environment with a single Franka Panda arm. Its 16 tasks span Counting, Permanence, Reference, and Imitation, requiring interaction history to track progress and guide subsequent actions. We examine whether explicit semantic modulation improves TTT's use of this history.

**Data and training.** A trained RoboTTT baseline generates rollouts on training seeds disjoint from evaluation. Human experts or autonomous expert agents determine DAgger intervention timing and corrective actions, yielding failure-to-correction trajectories. Semantic signals are computed from environment states during collection and online evaluation. Our policies and RoboTTT baselines use matched data and training protocols (Appendices D.1 and E.1).

**Baselines.** The benchmark's $\pi_{0.5}$ (Dai et al., 2026) initializes our TTT-augmented VLA model and provides a reference without explicit history modeling. TTT+Context (Dai et al., 2026), TTT+Modul (Dai et al., 2026), and TTT+Expert (Dai et al., 2026) compare TTT integration strategies, while FrameSamp+Modul (Dai et al., 2026) is the benchmark's strongest non-oracle method. External memory-based policies include MemER (Sridhar et al., 2025), which combines visual history with symbolic subgoals, SAM2Act+ (Fang et al., 2025), which maintains visual memory, and MemoryVLA (Shi et al., 2025), which combines perceptual and cognitive memory. RoboTTT (base) provides the reference, RoboTTT (fair) controls for additional data and training, and RoboTTT + DAgger tests the benefit of explicit semantics beyond failure-to-correction supervision.

All three SELA-TTT variants outperform RoboTTT (fair) and RoboTTT + DAgger on average (Figure 3). Subtask advancement is strongest at 55.96%, while overall goal progress leads on several counting and imitation tasks. These gains support explicitly modulating contextual contributions with task semantics beyond additional training and correction data. We next isolate the injection interface's contribution.

**Semantic injection interfaces.** Equivalent linear-attention injection outperforms input-feature and raw-QKV conditioning across all three semantic signals under matched conditions (Table 16, Appendix E.1.1), supporting this interface beyond semantic input alone.

**Semantic–context correspondence.** To test whether semantics shape the persistent TTT context and whether their effect depends on correspondence with recorded interactions, we perturb prior subtask-advancement labels. Altering labels tests their lasting influence beyond current feedback,

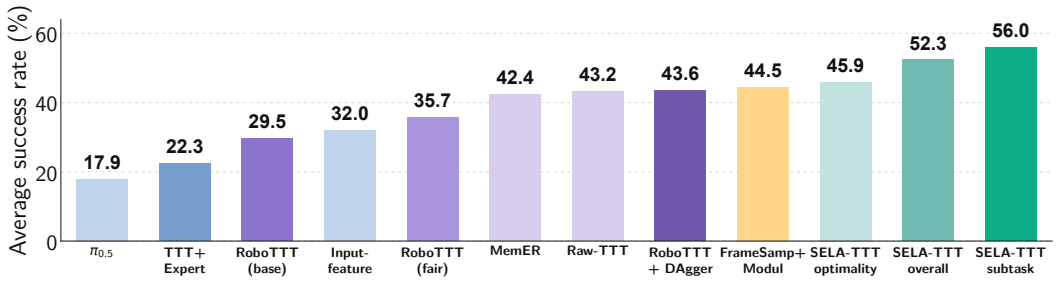



Figure 3: RoboMME success (%) across four task suites. RoboTTT (base) is the initialization reference; fair and DAgger use matched post-training. SELA-TTT uses subtask advancement, overall goal progress, or atomic-skill optimality.

while exchanging labels tests the importance of assigning semantic information to the corresponding interaction segments.

We retain each online continuation as Original. For Altered, we replay its recorded observations and actions from the pre-interval fast state with modified labels, then resume from the same environment state under unchanged current feedback (Table 1; Appendix F.1.1). Reduced success supports the lasting influence of semantic modulation through the fast state. Sensitivity to label exchanges further shows that semantic information must correspond to the interactions whose contributions it modifies, rather than act only as current feedback.

Table 1: Semantic–context correspondence (%; 50 continuations per task).

| | Success ↑ | | Targeted error ↓ | |
|---|---|---|---|---|
| Task (change) | Orig. | Alt. | Orig. | Alt. |
| PickHighlight ($-1 \rightarrow 0$) | 42 | 24 | 50 | 68 |
| PickXtimes ($+1 \rightarrow 0$) | 90 | 26 | 10 | 60 |
| SwingXtimes ($-1 \rightarrow +1$) | 78 | 14 | 16 | 74 |
| ButtonUnmaskSwap ($-1 \leftrightarrow +1$) | 36 | 6 | 64 | 84 |

Further ablations on read–write modulation, register count, normalization, inner learning rate and residual gating, gradient-truncation length, fast-model simplification, and context-length curriculum appear in Appendices E.1.2, C.2, C.3, C.4, C.5, C.6, and E.1.3, respectively.

### 5.2 Experiments on RMBench

**Benchmark.** RMBench (Chen et al., 2026) evaluates nine context-dependent tasks in a SAPIEN-based (Xiang et al., 2020) simulated tabletop environment with an ALOHA-AgileX dual-arm robot. These tasks require one or multiple past observations, covering different memory demands.

**Data and training.** We follow the matched protocols in Section 5.1, with benchmark-specific data and training detailed in Appendices D.2 and E.2.

**Baselines.** Alongside the same RoboTTT baselines, $\pi_{0.5}$ (Chen et al., 2026) and X-VLA (Zheng et al., 2025) provide references without explicit history modeling. Memory-based comparisons include subtask planning in Mem-0 (Chen et al., 2026), anchor frames and compressed history in MemoryWAM (Yang et al., 2026b), visual-anchor memory in EventVLA (Yang et al., 2026a), and historical-event memory with progress supervision in DiM-WAM (Wang et al., 2026a). Figure 4 compares average success across tasks for all three SELA-TTT signals.

All three SELA-TTT variants outperform RoboTTT (fair) and RoboTTT + DAgger on average; subtask advancement is strongest among these variants, at 80.89%. Overall goal progress and atomic-skill optimality remain close and lead or tie on several tasks (Table 26, Appendix F.2), supporting semantic modulation across memory demands.

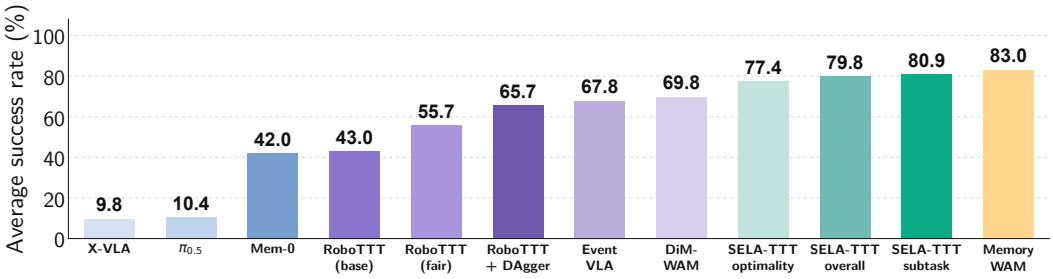



Figure 4: RMBench success (%) across nine tasks.

### 5.3 Real-World Experiments

**Task suites.** We evaluate our method on three task suites in a tabletop environment with a dual-arm AgileX Piper setup (Figure 5). See Appendix D.3 for setup and task details, and Figure 10 for annotated rollouts.

- **Repeated Picking.** The robot lifts a block of the instructed color and returns it to its original position for the requested number of cycles, then presses a button. The repeated scene requires context to track completed cycles.
- **Repeated Swinging.** The robot lifts the instructed block and moves it between two targets for the requested number of round trips, then returns it to its original position and presses a button. Execution requires tracking both movement phase and completed trips.
- **Chemistry Experiment.** Instructions specify the order of large-beaker placement on the stirrer, pouring from the medium and small beakers into it, stirrer activation, and one adjustment stage of one to three knob turns. Minimal visible traces of completed actions require context to track stages and counts.



Figure 5: Real-world setup.

**Data and training.** We collect expert demonstrations and DAgger failure-to-correction trajectories with semantic annotations. Using subtask advancement, SELA-TTT follows matched data and training protocols with RoboTTT baselines (Appendices D.3 and E.3), with a human operator supplying its semantic signals online.

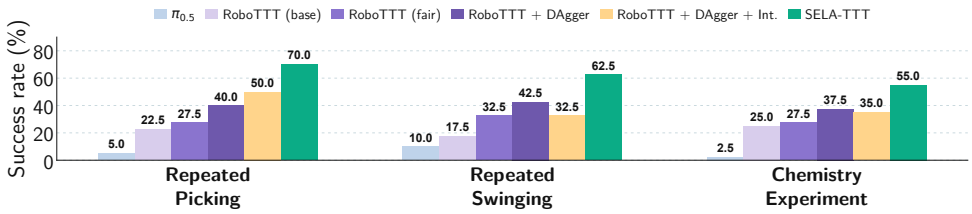



Figure 6: Real-world success (%; 40 episodes per suite).

**Baselines.** We compare $\pi_{0.5}$, RoboTTT (base), RoboTTT (fair), and RoboTTT + DAgger, alongside RoboTTT + DAgger + Int., which additionally receives test-time expert intervention that restores the pre-error subtask state while preserving completed progress (Appendix F.3).

**Task performance.** SELA-TTT achieves 62.50% average success across the three suites, exceeding RoboTTT (fair) by 33.33 percentage points under matched training resources (Figure 6). Gains over RoboTTT + DAgger further support explicitly incorporating task semantics into context updates beyond failure-to-correction supervision.

### 5.3.1 SEMANTIC-AWARE IN-CONTEXT LEARNING

We further use the Chemistry Experiment suite to test SELA-TTT's ability to learn from imperfect demonstrations, which we refer to as semantic-aware in-context learning. To ensure a fair evaluation, the post-trained variants receive the same mixture of perfect and imperfect human demonstrations paired with successful robot trajectories, with semantic annotations additionally provided to SELA-TTT.

Table 2: Semantic-aware ICL ($n = 20$).

| Method | Success (%) ↑ | Errors ↓ |
|---|---|---|
| RoboTTT (base) | 15 | 7 |
| RoboTTT (fair) | 35 | 6 |
| RoboTTT + DAgger | 45 | 6 |
| RoboTTT + DAgger + Int. | 45 | 7 |
| **SELA-TTT** | **65** | **4** |

Using each method's task-performance checkpoint, we evaluate one episode for each of 20 prompts paired with unseen imperfect demonstrations shared across methods (Appendix F.3.1). SELA-TTT achieves 65% success and reproduces undesired demonstrated behavior in fewer episodes than the RoboTTT baselines (Table 2), supporting selective use of unseen imperfect demonstrations during execution.

### 5.3.2 GENERALIZATION TO UNSEEN FAILURES

Using each method's task-performance checkpoint, we test semantically guided behavior under three failure conditions absent from training. Physical failure introduces an unseen interaction, while switching and continuation introduce unseen correspondences between interactions and negative semantic signals (Figure 11, Appendix F.3).

**Unseen Physical Failures.** We use the Repeated Picking suite to test generalization to unseen physical failures by manually releasing a fully lifted block so that it falls within reach. This interaction is unseen during training, and a negative semantic signal is provided afterward. The policy must pick it up again and complete the remaining instructed pick-and-place cycles (Appendix F.3.2).

**Unseen Switching Failures.** We use the Repeated Picking suite to test generalization to unseen switching failures. Approaching the first instruction-compatible block selected by the robot receives a negative semantic signal throughout the episode. The policy must switch to another matching block and complete the requested repetitions (Appendix F.3.3).

**Unseen Continuation Failures.** We use the Repeated Swinging suite to test generalization to unseen continuation failures. For instructions requiring multiple round trips, a negative signal midway through an outward or return movement requires stopping, putting down the block, and pressing the button. Completing this stop–release–button sequence defines success (Appendix F.3.3).

SELA-TTT achieves the highest success across all three tests, outperforming RoboTTT + DAgger with and without test-time expert intervention (Table 3). These results show that semantic modulation supports recovery, target switching, and suppression of undesired continuation when either the physical interaction or its correspondence with semantic feedback is unseen during training.

Table 3: Unseen failures: success (%; $n = 20$ per test). Bold marks the best results.

| Method | Physical | Switching | Continuation |
|---|---|---|---|
| RoboTTT + DAgger | 10 | 0 | 0 |
| RoboTTT + DAgger + Int. | 30 | 15 | 25 |
| **SELA-TTT** | **50** | **50** | **45** |

## 6 CONCLUSION

We introduce SELA-TTT to incorporate explicit task semantics into TTT through its equivalent linear-attention formulation. By modulating the queries, keys, and values governing context writes and reads, our method relates the contributions of interaction history to task progress and execution validity. Across simulation and real-world evaluations, this approach improves task performance and supports semantic-aware in-context learning from imperfect human demonstrations and generalization to previously unseen failures.

## AI USE STATEMENT

In this work, we used generative AI tools to assist with translation, method implementation, and dataset cleaning and reformatting. We did not use generative AI tools to propose or refine hypotheses, design research methodology or experiments or provide feedback on them, support qualitative or thematic data analysis, or interpret results. Formulating mathematical claims, generating synthetic datasets, developing theoretical models or conceptual frameworks, providing key ingredients for mathematical proofs, and assisting with proof writing were not applicable to this work. We also used generative AI tools to summarize and analyze existing literature, improve the readability of the manuscript, and identify relevant literature. All AI-assisted text was carefully reviewed and revised by the authors to ensure accuracy and consistency with the underlying research. The authors take full responsibility for the final content of this work.

## REFERENCES

Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. ATLAS: Learning to Optimally Memorize the Context at Test Time. *CoRR*, 2025a. URL https://doi.org/10.48550/arXiv.2505.23735.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to Memorize at Test Time. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/a4ca07aa108036f80cbb5b82285fd4b1-Abstract-Conference.html.

Tianxing Chen, Yuran Wang, Mingleyang Li, Yan Qin, Hao Shi, Zixuan Li, Yifan Hu, Yingsheng Zhang, Kaixuan Wang, Yue Chen, Hongcheng Wang, Renjing Xu, Ruihai Wu, Yao Mu, Yaodong Yang, Hao Dong, and Ping Luo. RMBench: Memory-Dependent Robotic Manipulation Benchmark with Insights into Policy Design. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2603.01229.

Egor Cherepanov, Nikita Kachaev, Daniil Zelezetsky, Aydar Bulatov, Artem Pshenitsyn, Yuri Kuratov, Alexey Skrynnik, Aleksandr I. Panov, and Alexey K. Kovalev. $\mu$VLA: On Recurrent Memory for Partially Observable Manipulation in VLA Models. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2606.12497.

Yinpei Dai, Hongze Fu, Jayjun Lee, Yuejiang Liu, Haoran Zhang, Jianing Yang, Chelsea Finn, Nima Fazeli, and Joyce Chai. RoboMME: Benchmarking and Understanding Memory for Robotic Generalist Policies. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2603.04639.

Karan Dalal, Daniel Koceja, Jiarui Xu, Yue Zhao, Shihao Han, Ka Chun Cheung, Jan Kautz, Yejin Choi, Yu Sun, and Xiaolong Wang. One-Minute Video Generation with Test-Time Training. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2025, Nashville, TN, USA, June 11-15, 2025*, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Dalal_One-Minute_Video_Generation_with_Test-Time_Training_CVPR_2025_paper.html.

Yiguo Fan, Pengxiang Ding, Shuanghao Bai, Xinyang Tong, Yuyang Zhu, Hongchao Lu, Fengqi Dai, Wei Zhao, Yang Liu, Siteng Huang, Zhaoxin Fan, Badong Chen, and Donglin Wang. Long-VLA: Unleashing Long-Horizon Capability of Vision Language Action Model for Robot Manipulation. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2508.19958.

Haoquan Fang, Markus Grotz, Wilbert Pumacay, Yi Ru Wang, Dieter Fox, Ranjay Krishna, and Jiafei Duan. SAM2Act: Integrating Visual Foundation Model with A Memory Architecture for Robotic Manipulation. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, 2025. URL https://proceedings.mlr.press/v267/fang25c.html.

Yusen Feng, Bingchen Han, Jiangran Lyu, Kai Liu, Yixin Zheng, Yuxuan Wan, Weiheng Liu, Sun Han, Ruiqin Li, Yulong Zhang, Fangfu Liu, Xuesong Shi, Libin Liu, Yizhou Wang, Zhizheng Zhang, and He Wang. WAM-TTT: Steering World-Action Models by Watching Human Play at Test Time. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2607.06988.

Juntao Gao, Feiyang Ye, Jing Zhang, and Wenjing Qian. Compressor-VLA: Instruction-Guided Visual Token Compression for Efficient Robotic Manipulation. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2511.18950.

Yihuai Gao, Jinyun Liu, Shuang Li, and Shuran Song. Gated Memory Policy. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2604.18933.

Youqiang Gui, Yuxuan Zhou, Shen Cheng, Xinyang Yuan, Haoqiang Fan, Peng Cheng, and Shuaicheng Liu. SeedPolicy: Horizon Scaling via Self-Evolving Diffusion Policy for Robot Manipulation. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2603.05117.

Xinying Guo, Chenxi Jiang, Hyun Bin Kim, Ying Sun, Yang Xiao, Yuhang Han, and Jianfei Yang. Chameleon: Episodic Memory for Long-Horizon Robotic Manipulation. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2603.24576.

Vidhi Jain, Maria Attarian, Nikhil J. Joshi, Ayzaan Wahid, Danny Driess, Quan Vuong, Pannag R. Sanketi, Pierre Sermanet, Stefan Welker, Christine Chan, Igor Gilitschenski, Yonatan Bisk, and Debidatta Dwibedi. Vid2Robot: End-to-End Video-Conditioned Policy Learning with Cross-Attention Transformers. In *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024. URL https://doi.org/10.15607/RSS.2024.XX.052.

Huiwon Jang, Sihyun Yu, Heeseung Kwon, Hojin Jeon, Younggyo Seo, and Jinwoo Shin. ContextVLA: Vision-Language-Action Model with Amortized Multi-Frame Context. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2510.04246.

Yunfan Jiang, Yevgen Chebotar, Ruijie Zheng, Fengyuan Hu, Yunhao Ge, Jimmy Wu, Tianyuan Dai, Scott Reed, Li Fei-Fei, Yuke Zhu, and Linxi Jim Fan. RoboTTT: Context Scaling for Robot Policies. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2607.15275.

JoyAI-RA Team. JoyAI-RA 0.5: Scaling Robot Manipulation Learning via Dual Action Alignment. *CoRR*, 2026. URL https://doi.org/10.48550/ARXIV.2608.05674.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers Are RNNs: Fast Autoregressive Transformers with Linear Attention. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, 2020. URL http://proceedings.mlr.press/v119/katharopoulos20a.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. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2510.00695.

Yuheng Lei, Zhixuan Liang, Hongyuan Zhang, and Ping Luo. VPWEM: Non-Markovian Visuomotor Policy with Working and Episodic Memory. *IEEE Robotics Autom. Lett.*, 2026. URL https://doi.org/10.1109/LRA.2026.3723315.

Hang Li, Fengyi Shen, Dong Chen, Liudi Yang, Xudong Wang, Jinkui Shi, Zhenshan Bing, Ziyuan Liu, and Alois Knoll. ReMem-VLA: Empowering Vision-Language-Action Model with Memory via Dual-Level Recurrent Queries. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2603.12942.

Yishu Li, Xinyi Mao, Ying Yuan, Kyutae Sim, Ben Eisner, and David Held. Learn from What We HAVE: History-Aware VErifier that Reasons about Past Interactions Online. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2509.00271.

Yushen Liang, Yue Peng, Baosheng Jin, Tianluo Zhang, Xinyu Zhang, Shuyi Zhou, Zhuoran Chen, Xinqi Liu, and Shenji Wan. TFP: Temporally Conditioned Memory-Fusion Policies for Visuomotor Learning. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2607.08283.

Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and Qiang Liu. Longhorn: State Space Models are Amortized Online Learners. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025. URL https://openreview.net/forum?id=8jOqCcLzeO.

Junchen Liu, Sven Elflein, Or Litany, Zan Gojcic, and Ruilong Li. Test-Time Training with KV Binding Is Secretly Linear Attention. *CoRR*, 2026a. URL https://doi.org/10.48550/arXiv.2602.21204.

Yuanzhe Liu, Jingyuan Zhu, Yuchen Mo, Gen Li, Xu Cao, Jin Jin, Yifan Shen, Zhengyuan Li, Tianjiao Yu, Wenzhen Yuan, Fangqiang Ding, and Ismini Lourentzou. PALM: Progress-Aware Policy Learning via Affordance Reasoning for Long-Horizon Robotic Manipulation. *CoRR*, 2026b. URL https://doi.org/10.48550/arXiv.2601.07060.

Max Sobol Mark, Jacky Liang, Maria Attarian, Chuyuan Fu, Debidatta Dwibedi, Dhruv Shah, and Aviral Kumar. BPP: long-context robot imitation learning by focusing on key history frames. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2602.15010.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 2018. URL https://doi.org/10.1609/aaai.v32i1.11671.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A Vision-Language-Action Model with Open-World Generalization. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2504.16054.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear Transformers Are Secretly Fast Weight Programmers. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, 2021. URL http://proceedings.mlr.press/v139/schlag21a.html.

Rutav Shah, Yisu Li, Femi Bello, Yuke Zhu, and Roberto Martín-Martín. Memory Retrieval in Visuomotor Policies for Long-Horizon Robot Control. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2606.25136.

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. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2508.19236.

Ajay Sridhar, Jennifer Pan, Satvik Sharma, and Chelsea Finn. MemER: Scaling Up Memory for Robot Control via Experience Retrieval. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2510.20328.

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. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, 2025. URL https://proceedings.mlr.press/v267/sun25h.html.

Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin, Jed McCaleb, Yejin Choi, and Yu Sun. End-to-End Test-Time Training for Long Context. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2512.23675.

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. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2603.03596.

Chen Wang, Linxi Fan, Jiankai Sun, Ruohan Zhang, Li Fei-Fei, Danfei Xu, Yuke Zhu, and Anima Anandkumar. MimicPlay: Long-Horizon Imitation Learning by Watching Human Play. In *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, 2023. URL https://proceedings.mlr.press/v229/wang23a.html.

Kai Wang, Zhaopeng Gu, Yixiang Chen, Yuan Xu, Qisen Ma, Peng Su, Zhaowen Li, Yan Huang, and Liang Wang. DIM-WAM: world-action modeling with diverse historical event memory. *CoRR*, 2026a. URL https://doi.org/10.48550/arXiv.2606.27677.

Ke Alexander Wang, Jiaxin Shi, and Emily B. Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *CoRR*, abs/2501.12352, 2025. URL https://doi.org/10.48550/arXiv.2501.12352.

Ziye Wang, Modi Shi, Chaojun Ni, Jiazhi Yang, Mengdi Li, Zhizhong Su, Tianwei Lin, and Hongyang Li. NativeMEM: Native Memory Compression for Long-Horizon Robotic Manipulation. *CoRR*, 2026b. URL https://doi.org/10.48550/arXiv.2607.06678.

Fanbo Xiang, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su. SAPIEN: A simulated part-based interactive environment. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, 2020. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Xiang_SAPIEN_A_SimulAted_Part-Based_Interactive_ENvironment_CVPR_2020_paper.html.

Hongyu Yan, Qiwei Li, Jiaolong Yang, and Yadong Mu. ProgressVLA: Progress-Guided Diffusion Policy for Vision-Language Robotic Manipulation. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2603.27670.

Ganlin Yang, Zhangzheng Tu, Yuqiang Yang, Sitong Mao, Junyi Dong, Tianxing Chen, Jiaqi Peng, Jing Xiong, Jiafei Cao, Jifeng Dai, Wengang Zhou, Yao Mu, and Tai Wang. EventVLA: Event-Driven Visual Evidence Memory for Long-Horizon Vision-Language-Action Policies. *CoRR*, 2026a. URL https://doi.org/10.48550/arXiv.2606.20092.

Sizhe Yang, Juncheng Mu, Tianming Wei, Chenhao Lu, Xiaofan Li, Linning Xu, Zhengrong Xue, Zhecheng Yuan, Dahua Lin, Jiangmiao Pang, and Huazhe Xu. MemoryWAM: Efficient World Action Modeling with Persistent Memory. *CoRR*, 2026b. URL https://doi.org/10.48550/arXiv.2606.20562.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated Delta Networks: Improving Mamba2 with Delta Rule. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, 2025. URL https://openreview.net/forum?id=r8H7xhYPwz.

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. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2505.23884.

Jinliang Zheng, Jianxiong Li, Zhihao Wang, Dongxiu Liu, Xirui Kang, Yuchun Feng, Yinan Zheng, Jiayin Zou, Yilun Chen, Jia Zeng, Ya-Qin Zhang, Jiangmiao Pang, Jingjing Liu, Tai Wang, and Xianyuan Zhan. X-VLA: Soft-Prompted Transformer as Scalable Cross-Embodiment Vision-Language-Action Model. *CoRR*, 2025. URL https://doi.org/10.48550/arXiv.2510.10274.

Yulin Zhou, Yuankai Lin, Fanzhe Peng, Jiahui Chen, Kaiji Huang, Hua Yang, and Zhouping Yin. MTIL: Encoding Full History With Mamba for Temporal Imitation Learning. *IEEE Robotics and Automation Letters*, 2025. URL https://doi.org/10.1109/LRA.2025.3615520.

Zheyu Zhuang, Ruiyu Wang, Nick Heppert, Johannes Fabian Hahn, Abhinav Valada, Florian T. Pokorny, and Danica Kragic. Attention from Action, for Action: Emergent Visual Bottlenecks for Policy Learning. *CoRR*, 2026. URL https://doi.org/10.48550/arXiv.2608.13422.

# A TTT Is Not a Raw Key–Value Retrieval System

Following Liu et al. (2026a), we examine whether TTT operates as a raw key–value retrieval system through three diagnostic experiments.

## A.1 Lower Inner Loss Does Not Imply Better Control

If TTT retrieves stored raw key–value associations, reducing the binding loss should improve their fidelity and thus benefit downstream performance. We test this prediction by varying the number of inner gradient-descent updates at inference while fixing the checkpoint and learning rate. Each variant is evaluated on the same 16 RoboMME tasks with ten episodes per task. For each inner-update variant, we sample four episodes per task and measure the final masked KVB loss at 64 decision boundaries, each corresponding to the first policy decision after context prefill. Sampling denoising evaluations 0, 3, 6, and 9 across all 18 TTT layers yields 4,608 loss measurements. Loss measurements use each variant's own rollouts, while downstream success is evaluated with paired episode seeds. Figure 7 compares the median final masked KVB loss over these measurements with closed-loop success. More updates reduce the median binding loss, yet success falls from 28.75% with one update to 5.00% with eight. The first four configurations produce no nonfinite policy outputs, so this decline is not explained by numerical failure. At 16, 32, and 64 updates, respectively, 2, 2, and 5 rollouts become nonfinite and are counted as failures. This inverse relationship contradicts the expected benefit of more accurate key–value storage.

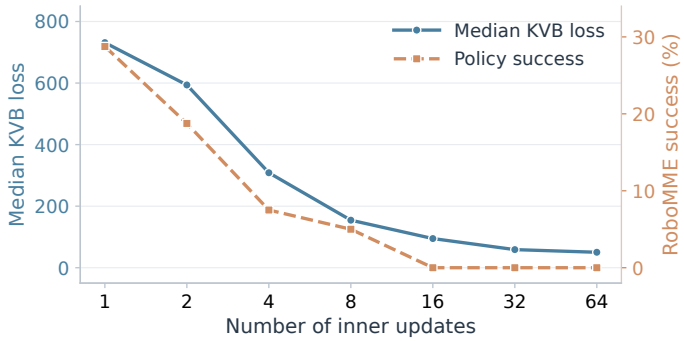



Figure 7: Increasing inner updates reduces median KVB loss (blue, left axis) but also lowers RoboMME success (orange, right axis). Success is measured over 160 paired episodes, counting nonfinite rollouts as failures.

## A.2 Distributional Separation of Raw Queries and Keys

For retrieval through a model fitted on keys, queries should substantially overlap with the key distribution so that the learned mapping remains applicable. We examine this expectation by jointly visualizing raw queries and keys in six layers sampled uniformly across the network depth (Figure 8).

We use task-balanced samples, L2 normalization, PCA to 50 dimensions, and t-SNE with perplexity 30, PCA initialization, 2,000 iterations, and seed 7. Queries and keys instead form visibly separated distributions. A query-versus-key classifier evaluated on held-out tasks achieves a mean AUC of 0.9996 across all 18 layers, providing a quantitative check outside the two-dimensional visualization. This separation reflects distributional asymmetry within a shared feature space. Evaluating the key-fitted model on differently distributed queries therefore challenges a direct interpolation-based retrieval interpretation.

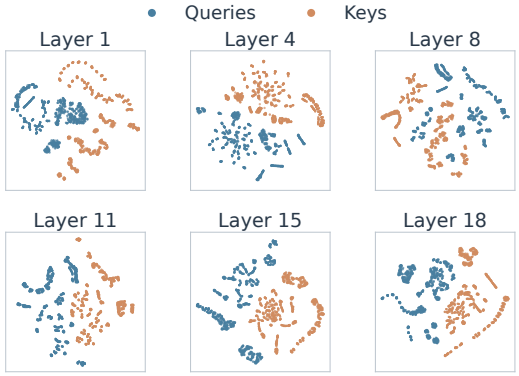



Figure 8: Joint Q/K t-SNE in six TTT layers. Each panel is fitted independently, so distances across panels are not comparable.

We additionally replace queries with keys when reading the updated fast model, leaving the binding update unchanged. Success remains essentially unchanged (Table 4), further challenging a raw key–value retrieval interpretation.

Table 4: RoboMME success rates (%) with query- or key-based readout over 50 episodes per task.

| Readout input | Counting | Permanence | Reference | Imitation | Average |
|---|---|---|---|---|---|
| Query | 44.50 | 23.00 | 28.00 | 23.50 | 29.75 |
| Key | 42.00 | 20.50 | 30.00 | 23.50 | 29.00 |

### A.3 Gradient Descent versus Gradient Ascent

If the inner loop stores key–value associations by fitting the binding objective, gradient ascent should impair retrieval by worsening that fit. We test this prediction by comparing policies trained with $\theta_{t+1} = \theta_t - \eta_t \nabla \ell_t^{\mathrm{KVB}}$ and $\theta_{t+1} = \theta_t + \eta_t \nabla \ell_t^{\mathrm{KVB}}$, while minimizing the same outer imitation objective. Each policy uses its respective inner rule throughout training and inference. The two policies achieve comparable success rates (Table 5), showing that effective context use does not depend on minimizing the binding loss to memorize key–value associations. This further demonstrates that our TTT layers do not operate as a raw key–value retrieval system.

Table 5: RoboMME success rates (%) over 50 episodes per task after training with inner gradient descent or ascent. Descent uses RoboTTT (base).

| Inner update | Counting | Permanence | Reference | Imitation | Average |
|---|---|---|---|---|---|
| Gradient descent | 44.50 | 23.00 | 28.00 | 23.50 | 29.75 |
| Gradient ascent | 41.50 | 23.50 | 31.50 | 22.00 | 29.63 |

## B TTT Is Equivalent to Linear Attention

### B.1 Derivation for a Biased Fast MLP

We specialize the sequence-level unrolling in Theorems 5.1–5.2 of Liu et al. (2026a) to our multi-token, write-then-read update. Their bias-free final layer is extended to the biased fast MLP used here by appending a constant feature. For one layer, one head, and one sequence, with the batch, layer, and head indices omitted,

$$\begin{aligned} &\phi_t(z) = \mathrm{GELU}(U_t^\top z + c_t), \quad f_t(z) = W_t^\top \phi_t(z) + b_t, \\ &\widetilde{\phi}_t(z) = \begin{bmatrix} \phi_t(z) \\ 1 \end{bmatrix}, \qquad \widetilde{W}_t = \begin{bmatrix} W_t \\ b_t^\top \end{bmatrix}, \qquad f_t(z) = \widetilde{W}_t^\top \widetilde{\phi}_t(z). \end{aligned} \tag{15}$$

Here $U_t \in \mathbb{R}^{d\times d_h}$ and $W_t \in \mathbb{R}^{d_h\times d_v}$ are the hidden and output weights, with biases $c_t \in \mathbb{R}^{d_h}$ and $b_t \in \mathbb{R}^{d_v}$. All fast parameters are updated by gradient descent with scalar step size $\eta_t$, without inner momentum or weight decay.

**Single-block update.** Let $m_{t,j} \in \{0,1\}$ mask token $j$ and $M_t = \sum_j m_{t,j} > 0$. For the implemented mean squared error, define the prediction gradient and effective value as

$$\begin{aligned} \ell_t^{\mathrm{KVB}} &= \frac{1}{M_t d_v}\sum_j m_{t,j}\,\|f_t(k_{t,j}) - v_{t,j}\|_2^2, \qquad g_{t,j} = \frac{\partial \ell_t^{\mathrm{KVB}}}{\partial f_t(k_{t,j})}, \\ \widehat{\mathbf{v}}_{t,j} &= -\eta_t g_{t,j} = \frac{2\eta_t m_{t,j}}{M_t d_v}\big(v_{t,j} - f_t(k_{t,j})\big). \end{aligned} \tag{16}$$

The factor $2/(M_t d_v)$ replaces the half-sum convention in Eq. 7. By the chain rule, the augmented affine state obeys

$$\begin{aligned} \nabla_{\widetilde{W}_t}\ell_t^{\mathrm{KVB}} &= \begin{bmatrix} \nabla_{W_t}\ell_t^{\mathrm{KVB}} \\ (\nabla_{b_t}\ell_t^{\mathrm{KVB}})^\top \end{bmatrix} = \sum_j \begin{bmatrix} \phi_t(k_{t,j}) \\ 1 \end{bmatrix} g_{t,j}^\top, \\ \widetilde{W}_{t+1} &= \widetilde{W}_t - \eta_t \sum_j \widetilde{\phi}_t(k_{t,j}) g_{t,j}^\top = \widetilde{W}_t + \sum_j \widetilde{k}_{t,j}\widehat{\mathbf{v}}_{t,j}^\top. \end{aligned} \tag{17}$$

Here $\widetilde{k}_{t,j} = [\phi_t(k_{t,j})^\top, 1]^\top$. The hidden parameters also update, yielding $\phi_{t+1}$ and the post-update query $\widetilde{q}_{t,\ell} = [\phi_{t+1}(q_{t,\ell})^\top, 1]^\top$. The one-block output is

$$\begin{aligned} y_{t,\ell} = f_{t+1}(q_{t,\ell}) &= \left(\widetilde{W}_t + \sum_j \widetilde{k}_{t,j}\widehat{\mathbf{v}}_{t,j}^\top\right)^\top \widetilde{q}_{t,\ell} \\ &= \widetilde{W}_t^\top \widetilde{q}_{t,\ell} + \sum_j \widehat{\mathbf{v}}_{t,j}(\widetilde{q}_{t,\ell}^\top \widetilde{k}_{t,j}). \end{aligned} \tag{18}$$

**Sequence-level unrolling.** The base case is $\widetilde{W}_1 = \widetilde{W}_0 + \sum_j \widetilde{k}_{0,j}\widehat{\mathbf{v}}_{0,j}^\top$. Assuming the expansion holds through block $t-1$, the next update gives

$$\begin{aligned} \widetilde{W}_{t+1} &= \widetilde{W}_t + \sum_j \widetilde{k}_{t,j}\widehat{\mathbf{v}}_{t,j}^\top \\ &= \widetilde{W}_0 + \sum_{i=0}^{t-1}\sum_j \widetilde{k}_{i,j}\widehat{\mathbf{v}}_{i,j}^\top + \sum_j \widetilde{k}_{t,j}\widehat{\mathbf{v}}_{t,j}^\top \\ &= \widetilde{W}_0 + \sum_{i=0}^{t}\sum_j \widetilde{k}_{i,j}\widehat{\mathbf{v}}_{i,j}^\top, \\ y_{t,\ell} = \widetilde{W}_{t+1}^\top \widetilde{q}_{t,\ell} &= \left(\widetilde{W}_0 + \sum_{i=0}^{t}\sum_j \widetilde{k}_{i,j}\widehat{\mathbf{v}}_{i,j}^\top\right)^\top \widetilde{q}_{t,\ell} \\ &= \widetilde{W}_0^\top \widetilde{q}_{t,\ell} + \sum_{i=0}^{t}\sum_j \widehat{\mathbf{v}}_{i,j}(\widetilde{q}_{t,\ell}^\top \widetilde{k}_{i,j}) \\ &= W_0^\top \phi_{t+1}(q_{t,\ell}) + b_0 + \sum_{i=0}^{t}\sum_j \widehat{\mathbf{v}}_{i,j}\left(\phi_{t+1}(q_{t,\ell})^\top \phi_i(k_{i,j}) + 1\right). \end{aligned} \tag{19}$$

This recovers Eq. 6, including the initial affine contribution and output-bias updates through $+1$. The hidden bias remains inside the evolving $\phi_t$. Entirely masked blocks skip the update and TTT residual. The identity is exact in real arithmetic and retains the nonlinear feature-map updates, without storing individual historical features.

### B.2 Numerical and Behavioral Equivalence

We compare the original fast-MLP forward with the equivalent linear-attention implementation using RoboTTT (base) without semantic modulation. The implementation keeps the learning-rate multiplication outside the contracted write and evaluates the bias separately, preserving the original floating-point operation order. The same-scene evaluation shares observations, diffusion noise, and batching while checking both paths' action chunks and complete fast states at every decision.

Across 16 RoboMME tasks with ten episodes each, the two implementations produce bitwise-identical action outputs and fast states at all 3,113 policy decisions, with the same 54 successful episodes (33.75%).

### B.3 Semantically Modulated Fast-State Recurrence

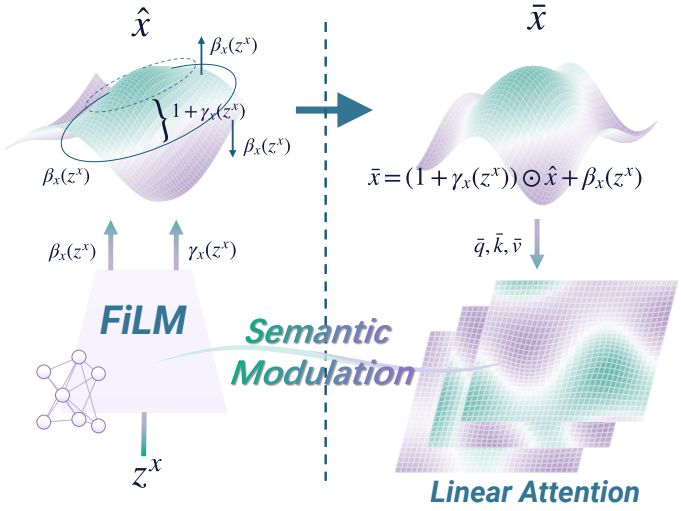



Figure 9: Semantic modulation of effective features.

For each block, evaluate the binding loss with the current fast model, update the parameters of $\phi_t$ by their KVB gradients, and form effective features as in Eq. 7. Reapplying the token mask after semantic modulation, with masked values set to zero, gives the affine write

$$\begin{aligned} W_{t+1} &= W_t + \sum_j \bar{k}_{t,j}\bar{v}_{t,j}^\top = W_0 + \sum_{i=0}^{t}\sum_j \bar{k}_{i,j}\bar{v}_{i,j}^\top, \\ b_{t+1} &= b_t + \sum_j \bar{v}_{t,j} = b_0 + \sum_{i=0}^{t}\sum_j \bar{v}_{i,j}, \\ y_{t,\ell} &= W_{t+1}^\top \bar{q}_{t,\ell} + b_{t+1} \\ &= W_0^\top \bar{q}_{t,\ell} + b_0 + \sum_{i=0}^{t}\sum_j \bar{v}_{i,j}\left(\bar{k}_{i,j}^\top \bar{q}_{t,\ell} + 1\right). \end{aligned} \tag{20}$$

Thus the same modulated query reads both the initial weights and accumulated updates. The next binding residual is evaluated with this updated fast model, not an independently maintained unmodulated model. Identity modulation recovers the original affine gradient step and readout. Otherwise, semantic modulation changes the affine update rule, while the linear-attention expansion remains exact for the resulting recurrence.

# C SELA-TTT ARCHITECTURE

**Register tokens.** Following RoboTTT (Jiang et al., 2026), learned registers collect visual-language information without passing the full prefix through TTT. We append them to the action tokens, allowing registers to attend to the prefix and actions while preserving the original action-token attention mask. Both token groups enter TTT and update its shared fast state, whereas only action positions receive outer action supervision.

**Integration and residual gating.** Each $\pi_{0.5}$ action-expert block applies joint attention, TTT on the resulting action and register features, and the original feed-forward sublayer. In the gated residual of Eq. 4, $Y_t$ stacks the outputs of Eq. 14. Initializing $g$ near zero limits the initial perturbation to the pretrained path while allowing the model to learn how strongly to incorporate context.

**Projections and positional encoding.** Let $\mathbf{x}_{t,j} \in \mathbb{R}^d$ be the column-vector representation of token $j$, whose transpose is the $j$th row of $X_t$. Biased linear projections produce queries, keys, and values, each followed by per-head L2 normalization and RoPE,

$$r^x_{t,j} = P_x \mathbf{x}_{t,j} + b_x, \qquad x_{t,j} = \underset{h=1}{\overset{N_h}{\mathrm{Concat}}} \mathrm{RoPE}_{p_{t,j}}\left(\frac{r^x_{t,j,h}}{\|r^x_{t,j,h}\|_2 + \epsilon}\right), \qquad x \in \{q,k,v\}. \tag{21}$$

Here $r^x_{t,j,h}$ is the $h$th of $N_h$ head slices, and $p_{t,j}$ advances across and within blocks. Each head is normalized and processed by its fast MLP before the head outputs are concatenated. We also evaluate the normalization strategies of LaCT (Zhang et al., 2025) and TTT-Video (Dalal et al., 2025) in Appendix C.3.

**Fast model and updates.** We construct the fast model as a biased two-layer MLP with a learned inner step size,

$$f_t(z) = W_t^\top \phi_t(z) + b_t, \qquad \eta_t = \alpha\,\mathrm{sigmoid}(\lambda). \tag{22}$$

Here $\alpha$ sets the step-size scale and $\lambda$ is learned through the outer objective. The parameters of $\phi_t$ follow the KVB gradient update, while Eq. 14 updates and reads the final affine layer, with the resulting parameters carried into the next chunk. Entirely masked blocks skip the update and residual contribution. At episode boundaries, all fast parameters return to their learned initialization.

**Semantic encoders and modulation.** Separate two-layer encoders at each TTT layer, with hidden width matching the fast MLP, map the selected semantic signal to branch-specific embeddings,

$$z^x_t = E_x(\rho) = B_x\,\mathrm{GELU}(A_x\rho + a_x) + b^{\mathrm{enc}}_x, \qquad x \in \{q,k,v\}. \tag{23}$$

Here $\rho$ follows the branch-specific timing in Section 4.2. The outer objective trains these encoders, whose outputs modulate effective features through FiLM. Embeddings are shared across tokens within each block, while the bias-carrying constant coordinate remains one. FiLM's scale and shift output heads are initialized to zero, with preceding weights randomly initialized, so that modulation initially preserves the effective features.

**Sequence action forcing and inference.** Following RoboTTT (Jiang et al., 2026), we sample noise and noise levels independently across successive action chunks during training,

$$\begin{aligned} \tau_t &\overset{\text{i.i.d.}}{\sim} p_\tau, & \varepsilon_t &\overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, I), \\ A^{\tau_t}_t &= (1-\tau_t)A_t + \tau_t\varepsilon_t, & u^{\mathrm{target}}_t &= \varepsilon_t - A_t. \end{aligned} \tag{24}$$

Here $p_\tau$ is the training noise-level distribution, and $u^{\mathrm{target}}_t$ is the flow-matching target. Separate masks control action supervision and state updates, allowing demonstration context to contribute without a robot-action target at every position. At inference, iterative denoising starts from Gaussian noise. All denoising evaluations for a chunk share the same incoming fast state, and only the final candidate updates are committed for the next chunk.

**Truncated backpropagation through time.** Following RoboTTT (Jiang et al., 2026), gradients propagate through inner updates within each segment of $B$ blocks. Writing all fast parameters as $\theta_t$, the incoming fast state is detached at subsequent segment boundaries,

$$\bar{\theta}_{mB} = \mathrm{sg}(\theta_{mB}), \qquad \mathrm{sg}(x) = x, \qquad \frac{\partial\,\mathrm{sg}(x)}{\partial x} = 0, \qquad m \geq 1. \tag{25}$$

The next segment starts from $\bar{\theta}_{mB}$, preserving accumulated context while truncating gradients across the boundary.

### C.1 Model Parameters

All RoboMME architecture ablations in this appendix use 50 episodes per task. Table 6 groups the standard model settings by component. The inner-learning-rate scale is $\alpha = 0.1$. Throughout the RoboMME comparisons, vanilla denotes RoboTTT (base), with 29.75% success (238/800). Register-count and learning-rate/gate sweeps are listed separately from this reference configuration in Appendices C.2 and C.4, with normalization comparisons in Appendix C.3.

Table 6: Standard model settings, grouped by component.

| Setting | Value |
|---|---|
| Architecture and context | |
| Action-expert hidden width | 1024 |
| Raw projection width | 1024 |
| Fast-MLP hidden width | 2048 |
| Fast-MLP output width | 1024 |
| Fast-MLP activation | GELU |
| TTT integration layers | 18 |
| Register tokens | 16 |
| Projection preprocessing | |
| Normalization heads | 4 |
| Normalization stabilizer | $10^{-5}$ |
| RoPE base | 10,000 |
| Fast-state updates and sequence training | |
| Inner updates per block | 1 |
| Inner-learning-rate scale $\alpha$ | 0.1 |
| Fast computation dtype | float32 |
| TBPTT segment length $B$ | 4 blocks |
| Initialization | |
| Learning-rate parameter $\lambda$ | 0 |
| Residual-gate parameter | 0.01 |
| Fast-weight standard deviation | 0.02 |
| Fast bias | 0 |

### C.2 Register-Count Ablation

We vary the register count $R \in \{16, 32, 64, 512\}$ while fixing the inner-learning-rate scale at $\alpha = 0.1$ and the initial residual gate at $g_0 = 0.001$, compared with $g_0 = 0.01$ in RoboTTT (base) (Table 7).

Table 7: Register-count sweep on RoboMME with fixed $\alpha = 0.1$ and $g_0 = 0.001$.

| Register tokens | 16 | 32 | 64 | 512 |
|---|---|---|---|---|
| Success (%) | 29.75 | 28.25 | 30.25 | 29.00 |

Increasing the register count yields fluctuating success rather than consistent gains. The limited benefit does not justify processing more tokens, so we retain 16 registers.

### C.3 Normalization Ablation

**Without normalization.** Removing the vanilla Q/K L2 normalization leads to numerical divergence rather than a usable policy.

**Reconstruction-target normalization.** Following TTT-Video (Dalal et al., 2025), we replace the MSE target by a normalized reconstruction of $v - k$. Within each head, let $\delta = v - k$ and $\mu(\delta)$ denote its channel mean,

$$v^{\text{norm}} = k + \gamma \odot \frac{\delta - \mu(\delta)}{s(\delta) + \epsilon} + \beta, \qquad s(\delta) = \sqrt{\frac{1}{d_{\text{head}} - 1} \sum_c (\delta_c - \mu(\delta))^2}. \tag{26}$$

Here $s(\delta)$ is the sample standard deviation over $d_{\text{head}}$ channels, $\epsilon = 10^{-8}$, and learned $(\gamma, \beta)$ start at $(1, 0)$. The remaining preprocessing, fast model, and update rule are unchanged.

**Fast-weight normalization.** Adapting the normalization in LaCT (Zhang et al., 2025), within each head we rescale each updated column of $F \in \{U, W\}$ to its pre-update norm,

$$F_t^{\text{cand}} = F_t - \eta_t \nabla_{F_t} \ell_t^{\text{KVB}}, \qquad F_{t+1,:,j} = \frac{\|F_{t,:,j}\|_2}{\|F_{t,:,j}^{\text{cand}}\|_2 + \epsilon} F_{t,:,j}^{\text{cand}}. \tag{27}$$

Here $\epsilon = 10^{-5}$. Only this rescaling is added to the MSE update, with the architecture, objective, and ordinary bias updates unchanged.

Table 8: Normalization ablation on RoboMME. Vanilla is RoboTTT (base).

| Variant | Success (%) |
|---|---|
| Vanilla (RoboTTT base) | 29.75 |
| TTT-Video target normalization | 24.25 |
| LaCT normalization only | 23.50 |
| Without Q/K normalization | Diverged |

Vanilla normalization achieves the highest success among the tested variants. It retains per-head L2 normalization of the projected inputs without adding reconstruction-residual normalization or fast-weight rescaling.

### C.4 Learning Rate and Residual Gate

At $R = 16$, we vary the inner-learning-rate scale $\alpha$ with $g_0 = 0.01$, and the initial residual gate $g_0$ with $\alpha = 0.1$. Table 9 lists these two partial sweeps, which examine fast-state adaptation and its contribution to the pretrained policy. Register-count variation is evaluated separately in Appendix C.2.

Table 9: Learning-rate and gate sweeps at $R = 16$. Each panel varies one parameter while fixing the other.

| Inner-rate sweep ($g_0 = 0.01$) | | Gate sweep ($\alpha = 0.1$) | |
|---|---|---|---|
| $\alpha$ | Success (%) | $g_0$ | Success (%) |
| 0.1 | 29.75 | 0.001 | 29.75 |
| 0.2 | 29.00 | 0.002 | 27.75 |
| 1.0 | 30.00 | 0.01 | 29.75 |

### C.5 Gradient-Truncation Ablation

We compare truncation lengths of four and eight blocks on RoboMME, keeping the forward context length and remaining training settings fixed. Eight blocks provide no improvement over four, while sixteen blocks exceed GPU memory capacity. We therefore use four blocks.

Table 10: TBPTT ablation on RoboMME.

| Truncation length | Training status | Success (%) |
|---|---|---|
| 4 blocks (RoboTTT base) | Completed | 29.75 |
| 8 blocks | Completed | 29.25 |

### C.6 Architectural Simplification

Following the simplification trajectory of TTT-is-LA (Liu et al., 2026a), we compare the adaptive fast MLP with a last-layer-only variant and a single linear fast model. The last-layer-only variant retains $f_t(z) = W_t^\top \phi(z) + b_t$, with $\phi$ fixed during inner updates but learned through the outer objective. The linear variant further removes $\phi$, giving $f_t(z) = W_t^\top z + b_t$. All variants retain the MSE binding objective and follow the same RoboMME training protocol.

Table 11: Architectural simplification on RoboMME. Success is averaged over all 16 tasks.

| Fast model | Inner-updated components | Success (%) ↑ |
|---|---|---|
| Adaptive MLP | $\phi_t, W_t, b_t$ | 29.75 |
| Last-layer-only MLP | $W_t, b_t$ | 28.50 |
| Single linear layer | $W_t, b_t$ | 24.25 |

The adaptive fast MLP achieves the highest success (Table 11), supporting its retention in our policy. We therefore use the equivalent linear-attention formulation to guide semantic modulation without adopting these architectural reductions from TTT-is-LA (Liu et al., 2026a).

## D Dataset and Split Protocol

Across the two simulation benchmarks, we first train RoboTTT (base) on standard expert demonstrations and use it to collect rollouts on training seeds. Expert interventions provide corresponding failure-to-correction trajectories, while motion planning generates additional expert demonstrations on the same seeds. Collection seeds are shared across methods and remain disjoint from evaluation seeds. For real-world experiments, human experts provide the demonstrations and corrective interventions, with additional expert trajectories collected under the corresponding prompts. Both settings provide expert demonstrations and failure-to-correction trajectories for matched comparisons.

Starting from RoboTTT (base), RoboTTT (fair) continues training on the additional expert demonstrations, whereas RoboTTT + DAgger and SELA-TTT use the same failure-to-correction trajectories. Chemistry Experiment additionally includes shared video-prompted expert demonstrations, as detailed in Appendix D.3. The additional datasets contain equal numbers of trajectory records, and the post-training variants receive the same number of updates within each benchmark or suite. On the simulation benchmarks, the three SELA-TTT variants share the dataset and differ only in semantic signal.

We calculate the three semantic signals from subtask progress and intervention records to capture unsuccessful execution, corrective behavior, and progress toward completion. Observation and demonstration viewing provide context rather than separate subtasks or primitives. During training replay and evaluation, robot-interaction signals condition only subsequent predictions and are supplied once the corresponding behavior is observable. Experiment-specific collection and signal calculations are detailed below.

## D.1 RoboMME

### D.1.1 Subtask Composition

Table 12 organizes each task into ordered, goal-directed subtasks. Approach, grasp, transport, and release belong to the same subtask when they serve one object-level objective. A change of manipulated object, destination, or required stage starts a new subtask, while repeated cycles instantiate the same subtask type. Each required repetition counts as a separate subtask instance for overall goal progress, while repeated instances share the same primitive identity for atomic-skill optimality. Environment predicates track their progress and completion.

Table 12: RoboMME subtask composition used for data collection. Repetition follows the instruction, and termination denotes the task's completion action where applicable.

| Category | Task | Subtask composition |
|---|---|---|
| Counting | BinFill | fill with one object → repeat → terminate |
| | StopCube | stop by pressing the button at the requested count |
| | PickXtimes | pick and place the selected block on the target → repeat → terminate |
| | SwingXtimes | move outward → return → repeat → put down/terminate |
| Permanence | ButtonUnmask | press → uncover each requested block in order |
| | VideoUnmask | uncover each requested block in order |
| | VideoUnmaskSwap | uncover tracked blocks after swapping |
| | ButtonUnmaskSwap | press both buttons → uncover tracked blocks after swapping |
| Reference | PickHighlight | press → lift and put down each highlighted block → terminate |
| | VideoRepick | re-pick and put down the indicated block → repeat/terminate |
| | VideoPlaceButton | place the block at the button-referenced target |
| | VideoPlaceOrder | place the block at the ordinally referenced target |
| Imitation | MoveCube | move the cube using the demonstrated technique |
| | InsertPeg | grasp the indicated peg end → insert from the indicated side |
| | PatternLock | retrace each segment with the attached stick |
| | RouteStick | navigate around each obstacle along the demonstrated path |

### D.1.2 Data Collection

We jointly train RoboTTT (base) using 100 expert demonstrations per task, then collect 200 failure-to-correction trajectories per task on training seeds. In 100 trajectories, the expert completes the remaining task after intervention. In the other 100, the expert corrects and completes the current subtask before returning control to the policy, with further corrections as needed until task completion. Intervention conditions are randomized and trajectories are filtered to diversify initial states, failures, and corrective behaviors. Motion planning additionally generates 200 expert trajectories per task on the same collection seeds.

RoboTTT (fair) uses these 200 additional expert trajectories, while RoboTTT + DAgger and SELA-TTT use all 200 corrected trajectories. All three methods continue training from the same base checkpoint for the same number of updates, and all collection seeds remain disjoint from evaluation seeds.

### D.1.3 Semantic Signal Assignment

At decision boundary $t$, let $\mathcal{S}_t$ record subtask progress, completed subtasks, and the primitive executed in the preceding interaction block. Combining this subtask record with expert intervention, $\mathcal{I}_t$ marks subtask deviation as $-1$, the correction interval before valid execution resumes as $0$, and valid subtask execution as $+1$.

**Subtask advancement.** We assign advancement labels using this subtask-informed intervention record,

$$\rho_t^{\mathrm{adv}} = \begin{cases} -1, & \mathcal{I}_t = -1, \\ 0, & \mathcal{I}_t = 0, \\ +1, & \mathcal{I}_t = +1. \end{cases} \tag{28}$$

This distinguishes subtask deviation, neutral correction, and valid progress.

**Overall goal progress.** We evaluate the completion conditions in $\mathcal{S}_t$ and recheck them during intervention intervals $\mathcal{I}_t = 0$, giving

$$\rho_t^{\text{goal}} = N - \#\{\text{completed subtasks in } \mathcal{S}_t\}. \tag{29}$$

Completion requires the subtask's completion condition rather than a positive block label, so retries do not create additional requirements.

**Atomic skill optimality.** We sign the active primitive recorded in $\mathcal{S}_t$ using the subtask–intervention label above,

$$\boldsymbol{\rho}_t^{\text{skill}} = \big(1 - 2\mathbf{1}[\rho_t^{\text{adv}} = -1]\big)\,\text{onehot}(\mathcal{S}_t). \tag{30}$$

Here $\text{onehot}(\mathcal{S}_t)$ selects the executed primitive, or is zero when none is active, distinguishing negatively evaluated execution from other execution in a common primitive space.

## D.2 RMBench

### D.2.1 Subtask Composition

RMBench provides action-aligned language annotations (Chen et al., 2026) and task-state checks, which support the subtask compositions in Table 13.

Table 13: RMBench subtask composition. $n$ denotes the required repetition count, with subscripts distinguishing button counts; repeated object operations follow the required order.

| Task | Subtask composition |
|---|---|
| Observe and Pick Up | Grasp and lift the matching object |
| Rearrange Blocks | Move middle block to pad → press button → move remaining block to center |
| Put Back Block | Move block to center → press button → return block to original pad |
| Swap Blocks | Move first block to spare pad → move second block to first block's pad → move first block to second block's pad → press button |
| Swap T | (Grasp T → reorient → place at exchanged pose) $\times 2$ |
| Blocks Ranking Try | Arrange the three blocks → press button → revise and repeat until correct |
| Press Button | (Press left button) $\times n_{\text{left}}$ → (press middle button) $\times n_{\text{middle}}$ → press right button to confirm |
| Cover Blocks | Cover blocks from left to right → uncover in red–green–blue order and return each cover |
| Battery Try | Insert both batteries with the required orientations → withdraw and retry unsuccessful insertions |

### D.2.2 Data Collection

We train RoboTTT (base) on the benchmark's 50 expert demonstrations per task (Chen et al., 2026), then collect 100 failure-to-correction trajectories per task. Half use full expert continuation and half return control after correction of the current subtask. Motion planning generates 100 additional expert trajectories per task on the same seeds. These train RoboTTT (fair), while all 100 corrected trajectories per task train RoboTTT + DAgger and SELA-TTT. Intervention randomization, filtering, and matched further training follow Appendix D.1.

Collection uses the same training seeds across methods, with an identical evaluation seed list held out from all training data. Corrections begin from the encountered failure state and preserve the completed subtasks and original instruction.

### D.2.3 Semantic Signal Assignment

The three signals follow Appendix D.1.3, using the subtask compositions in Table 13 and the intervention records.

### D.3 Real-World Experiments

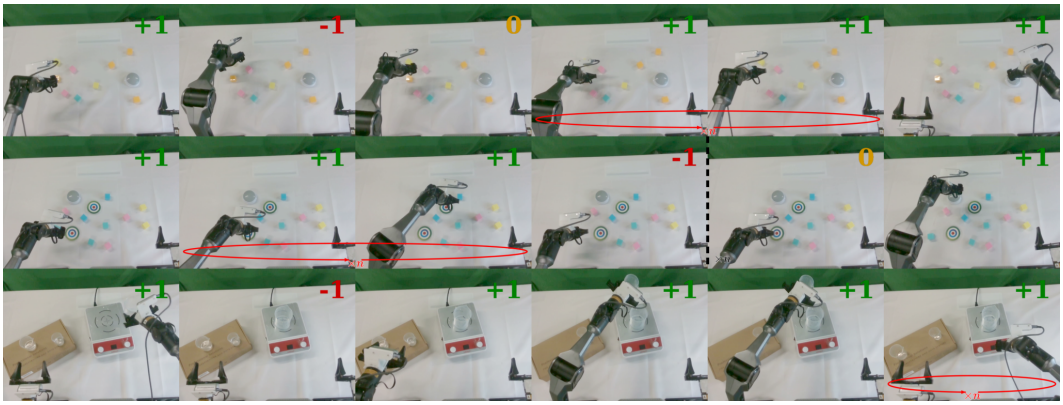



Figure 10: Annotated rollouts for the three real-world suites (top to bottom): Repeated Picking, Repeated Swinging, and Chemistry Experiment. Labels show subtask advancement; ellipses mark repetitions.

#### D.3.1 Subtask Composition

All three suites use the dual-arm AgileX Piper tabletop setup introduced in Section 5.3. Repeated Picking presents blocks of multiple colors and a button; blocks matching the instructed color are valid targets. There is no placement target, and four instructed colors combined with one to five repetitions give 20 prompts. Repeated Swinging adds two targets and combines four color-specific choices and one unrestricted choice with one to three round trips to give 15 prompts. Chemistry Experiment uses one large, medium, and small beaker and a magnetic stirrer with a switch and knob. Stage orderings are sampled subject to large-beaker placement preceding both pouring stages and stirrer activation preceding knob adjustment. Treating consecutive knob turns as one stage yields 20 valid orderings, each paired with one to three turns to give 60 prompts. Table 14 summarizes their subtask compositions.

Table 14: Real-world subtask composition. $n$ is the instructed repetition count; Chemistry Experiment stages follow the instructed order.

| Task suite | Subtask composition |
|---|---|
| Repeated Picking | Grasp block $\rightarrow$ (lift $\rightarrow$ lower to original position) $\times n \rightarrow$ press button |
| Repeated Swinging | Grasp block $\rightarrow$ lift $\rightarrow$ (move to first target $\rightarrow$ move to second target) $\times n \rightarrow$ return to original position and release $\rightarrow$ press button |
| Chemistry Experiment | Random order {place large beaker on stirrer; pour from medium beaker; pour from small beaker; activate stirrer; (turn knob) $\times n$} |

#### D.3.2 Data Collection

**Repeated Picking.** We collect 100 expert trajectories using ten prompts to train RoboTTT (base). Prompts are then sampled randomly from all 20 configurations for rollouts and expert correction, yielding 200 failure-to-correction trajectories.

**Repeated Swinging.** Following the same procedure, we collect 100 expert trajectories using ten prompts, then sample from all 15 configurations to obtain 200 failure-to-correction trajectories.

**Chemistry Experiment.** We collect 200 expert trajectories using 20 prompts. Among these, 50 robot trajectories are each paired with both a perfect and an imperfect human demonstration, yielding 100 demonstration–execution pairs from 50 distinct robot trajectories. Subsequent rollouts across all 60 configurations yield 200 failure-to-correction trajectories, including 50 collected under

human-video prompting. During post-training, RoboTTT (fair), RoboTTT + DAgger, and SELA-TTT also reuse the original demonstration–execution pairs alongside their respective additional datasets, with the same mixture of perfect and imperfect demonstrations and subtask-advancement annotations additionally provided to SELA-TTT. Imperfect demonstrations include an undesired subtask executed partway and then reversed, with the undesired execution labeled as failure and the rollback marked as intervention. Human demonstrations provide context, while robot action supervision follows the trajectory-specific masks. Video-conditioned episodes use an imitation instruction that asks the policy to reproduce the desired behavior in the demonstration, without specifying the procedure separately.

Across suites, expert continuations and matched further training follow Appendix D.1. For RoboTTT (fair), we collect additional expert trajectories under the corresponding rollout prompts and conditioning inputs, including 50 Chemistry Experiment trajectories under human-video prompting; the resulting sets contain 200 trajectories for each suite, matching the additional trajectory counts used by SELA-TTT. RoboTTT + DAgger and SELA-TTT use the same failure-to-correction trajectories. Training includes ordinary execution, wrong-object selections, and their corrections under the original instructions.

#### D.3.3 Semantic Signal Assignment

Signal definitions follow Appendix D.1.3, with human annotators assigning labels from subtask progress and intervention records for both robot trajectories and paired human demonstrations.

## E Training

The architecture follows Appendix C. We first train the added TTT layers with the pretrained backbone frozen, then unfreeze all model parameters for matched post-training, following the two-stage practice of RoboTTT (Jiang et al., 2026). Here $T$ counts successive blocks, and all batch sizes are global.

### E.1 RoboMME

We initialize from the officially released RoboMME $\pi_{0.5}$ baseline (Dai et al., 2026), trained jointly on all 16 tasks with the vision encoder frozen and no proprioceptive input. Table 15 records its published recipe alongside our TTT training settings.

**RoboTTT (base).** Keeping the backbone frozen, we train the TTT layers and register tokens on the 16-task expert mixture with $T = 1, 4, 8, 32, 64, 96$, using 10,000 updates per stage. The TTT learning rate increases linearly to $5 \times 10^{-5}$ over the first 10,000 updates and remains constant for the rest of training, without restarting warm-up at curriculum transitions. The policy predicts 20 actions and executes 16 before the next context update, so $T = 96$ spans 1,536 control steps, exceeding the benchmark's 1,300-step episode limit. The resulting model is RoboTTT (base). Active training requires 60 hours on eight H100 GPUs.

**Matched post-training.** Starting from this base, all post-training configurations train for 20,000 updates at $T = 96$. RoboTTT (fair) uses the additional expert trajectories, while RoboTTT + DAgger uses failure-to-correction trajectories. All valid blocks update the TTT fast state. Expert demonstrations provide action targets throughout, while failure-to-correction trajectories provide action targets only on expert corrections; within each model, the action-supervision mask applies to all trainable parameter groups. SELA-TTT adds semantic encoders and uses the same supervision mask. TTT and backbone learning rates remain constant at $5 \times 10^{-5}$ and $2.5 \times 10^{-6}$, respectively, keeping the backbone rate at one twentieth of the TTT rate without restarting warm-up. Newly added semantic encoders use a constant learning rate of $5 \times 10^{-5}$ without warm-up. Batch size remains 64 throughout the comparison (Table 15).

Table 15: RoboMME training settings, including the released baseline's recipe. Learning rates refer to outer optimization. All variants jointly train on 16 tasks.

| | | | | |
|---|---|---|---|---|
| Released $\pi_{0.5}$ baseline | | | | |
| Training updates | 80,000 | | | |
| Global batch size | 64 | | | |
| Action horizon | 20 | | | |
| Learning rate | $5 \times 10^{-5}$ | | | |
| Learning-rate schedule | Constant after warm-up | | | |
| Warm-up updates | 10,000 | | | |
| Optimizer | AdamW | | | |
| Adam first-moment decay | 0.9 | | | |
| Adam second-moment decay | 0.95 | | | |
| Weight decay | 0 | | | |
| Gradient clip | 1.0 | | | |
| EMA decay | 0.999 | | | |
| Frozen parameters | Vision encoder | | | |
| TTT training and matched post-training | | | | |
| Setting | RoboTTT (base) | RoboTTT (fair) | +DAgger | SELA-TTT |
| Initialization | Released $\pi_{0.5}$ | RoboTTT (base) | | |
| Training updates | $6 \times 10{,}000$ | 20,000 | 20,000 | 20,000 |
| Context length $T$ | $1, 4, 8, 32, 64, 96$ | 96 | 96 | 96 |
| Global batch size | 64 | 64 | 64 | 64 |
| Trainable parameters | TTT + registers | All | All | All + encoders |
| TTT peak learning rate | $5 \times 10^{-5}$ | | | |
| Backbone peak learning rate | Frozen | $2.5 \times 10^{-6}$ | | |
| Semantic-encoder learning rate | – | – | – | $5 \times 10^{-5}$ |
| Semantic-encoder warm-up updates | – | – | – | 0 |
| Semantic-encoder LR schedule | – | – | – | Constant |
| TTT-only warm-up | 10,000 | – | – | – |
| Existing-parameter post-training schedule | – | Constant | | |
| Existing-parameter post-training warm-up | – | 0 | | |
| Optimizer | AdamW | | | |
| Adam first-moment decay | 0.9 | | | |
| Adam second-moment decay | 0.95 | | | |
| Weight decay | $10^{-5}$ | | | |
| Gradient clip | 1.0 | | | |
| EMA decay | 0.999 | | | |
| Action horizon | 20 | | | |
| Executed prefix | 16 | | | |
| TBPTT segment length | 4 blocks | | | |
| Training hardware | 8× H100 | | | |

Full-parameter post-training at $T = 96$ requires 46 hours for 20,000 updates on eight H100 GPUs, excluding additional semantic-modulation overhead, compilation, checkpoint transfer, and interruptions.

### E.1.1 Semantic Injection Interfaces

Table 16: FiLM interfaces on RoboMME (success, %; 16 tasks). Bold marks the best results.

| Semantic signal | Input | Raw | Equiv. LA |
|---|---|---|---|
| Subtask advancement | 32.00 | 43.15 | **55.96** |
| Overall goal progress | 33.88 | 41.94 | **52.33** |
| Atomic skill optimality | 32.81 | 38.14 | **45.93** |

Equivalent linear-attention injection achieves higher average success across all three signals (Table 16). Because the compared interfaces receive the same semantic information, these results support

modulating the representations governing context writes and reads, rather than providing semantic input alone.

The interface comparison uses FiLM with each of the three semantic signals under these training settings. Input-feature FiLM conditions each post-attention action or register token $\mathbf{x}_{t,j}$ entering the TTT branch, where $\mathbf{x}_{t,j}^\top$ is the $j$th row of $X_t$,

$$z_t^{\text{in}} = E_{\text{in}}(\rho_t), \qquad \overline{\mathbf{x}}_{t,j} = (1 + \gamma_{\text{in}}(z_t^{\text{in}})) \odot \mathbf{x}_{t,j} + \beta_{\text{in}}(z_t^{\text{in}}). \tag{31}$$

The conditioned features precede QKV projection, normalization, and positional encoding, while the original residual stream is unchanged. For overall goal progress, we instead use $z_t^{\text{in}} = E_{\text{in}}([\rho_{t-1}^{\text{goal}}; \rho_t^{\text{goal}}])$, where the two signals are concatenated. Raw-QKV FiLM instead modulates the preprocessed queries, keys, and values immediately before the fast MLP. Both variants retain the original KVB update and use matched modulation settings (Table 17).

Table 17: Matched FiLM settings for the injection interfaces. Equivalent-LA settings are unchanged. A dash denotes a branch not separately modulated.

| Setting | Input features | Raw TTT | Equivalent LA |
|---|---|---|---|
| Input-feature head output width | 1024 | – | – |
| Query head output width | – | 1024 | 2048 |
| Key head output width | – | 1024 | 2048 |
| Value head output width | – | 1024 | 1024 |
| Encoder linear layers | 2 | 2 | 2 |
| Encoder hidden width | 1536 | 2720 | 2048 |
| Encoder activation | GELU | GELU | GELU |
| Input-feature embedding width | 8192 | – | – |
| Query embedding width | – | 2048 | 2048 |
| Key embedding width | – | 2048 | 2048 |
| Value embedding width | – | 2048 | 1024 |
| Training updates | 20,000 | 20,000 | 20,000 |
| Global batch size | 64 | 64 | 64 |

### E.1.2 Read–Write Modulation Ablation

To determine whether task semantics should guide context reads, writes, or both, we separately train Q-only, KV-only, and QKV variants using subtask advancement and the same post-training protocol. In Eq. 14, their reading and writing features are respectively

$$(q_t^\star, k_t^\star, v_t^\star) = \begin{cases} (\overline{q}_t, \widehat{k}_t, \widehat{v}_t), & \text{Q-only}, \\ (\widehat{q}_t, \overline{k}_t, \overline{v}_t), & \text{KV-only}, \\ (\overline{q}_t, \overline{k}_t, \overline{v}_t), & \text{QKV}. \end{cases} \tag{32}$$

These features replace the corresponding barred quantities in that equation, including the value used for the bias update. All variants retain the KVB updates to the hidden-representation parameters $\Theta_t$ in $\phi_t(\cdot) = \phi(\cdot; \Theta_t)$ and carry the resulting fast state across chunks. Action-supervision masks are identical across these semantic-modulation comparisons. The read–write comparison uses 450 episodes per task under the RoboMME evaluation procedure in Appendix F.1.

Table 18: Read–write ablation on RoboMME with 450 episodes per task. Success rates (%) by suite and overall.

| Variant | Count. | Perm. | Ref. | Imit. | Avg. |
|---|---|---|---|---|---|
| Q-only | 66.67 | 31.50 | 43.06 | 54.50 | 48.93 |
| KV-only | 64.22 | 34.94 | 41.06 | 52.83 | 48.26 |
| QKV | 70.94 | 43.61 | 47.17 | 62.11 | 55.96 |

The read-and-write QKV variant achieves the highest average success (Table 18), showing that conditioning both reads and writes supports more effective use of context than conditioning reading or writing alone.

### E.1.3 CONTEXT-LENGTH CURRICULUM ABLATION

We compare the main curriculum with a three-stage schedule and training at $T = 96$ throughout, allocating 60,000 updates to each and evaluating each configuration on 50 episodes per task. Table 19 compares success with total active training time under the frozen-backbone protocol, isolating the cost of exposing long context earlier in training.

Table 19: Context-length curricula on RoboMME. Active training times on eight H100 GPUs exclude compilation and interruptions.

| Context stages $T$ | Updates per stage | Success (%) | Training time (h) |
|---|---|---|---|
| $1 \to 4 \to 8 \to 32 \to 64 \to 96$ | 10,000 | 29.75 | 60 |
| $16 \to 64 \to 96$ | 20,000 | 29.25 | 95 |
| 96 throughout | 60,000 | 31.00 | 127 |

Training at $T = 96$ throughout yields only a minor gain of 1.25 percentage points while more than doubling training time, from 60 to 127 hours. The three-stage schedule also takes longer without improving success. We therefore retain the progressive curriculum used by RoboTTT (Jiang et al., 2026).

## E.2 RMBENCH

We train a separate policy for each of the nine tasks. Each $\pi_{0.5}$ baseline is reproduced from the pretrained $\pi_{0.5}$ model using 50 expert trajectories and the benchmark's released full-fine-tuning configuration (Chen et al., 2026). This configuration uses 20,000 updates, batch size 64, a 50-step action horizon, and proprioceptive state encoded in the prompt. Table 20 records its optimization settings and the subsequent TTT stages.

Each chunk executes all 50 predicted actions before the next context update. From each task-specific baseline, we train the TTT layers with the backbone frozen, following the ordered curriculum $T = 1, 4, 8, 32, 64, 96$ with 10,000 updates per stage and stopping at the first stage $T_{\max}$ for which $50T_{\max}$ exceeds the task's longest training episode in environment steps. Fair, DAgger, and SELA-TTT then train separately for 10,000 updates at $T_{\max}$. Their parameter groups, backbone-to-TTT learning-rate ratio, and DAgger supervision masks follow Appendix E.1, retaining batch size 64 across methods.

Table 20: RMBench training settings, applied independently to each task. $T_{\max}$ is determined from the task's training episodes.

| Setting | $\pi_{0.5}$ baseline | RoboTTT (base) | Post-training variants |
|---|---|---|---|
| Training updates | 20,000 | 10,000 per $T$ stage | 10,000 each |
| Global batch size | 64 | 64 | 64 |
| Trainable parameters | All | TTT + registers | All; encoders in SELA-TTT |
| Context length | – | Increasing to $T_{\max}$ | $T_{\max}$ |
| Action horizon | 50 | 50 | 50 |
| Baseline peak learning rate | $2.5 \times 10^{-5}$ | – | – |
| Baseline final learning rate | $2.5 \times 10^{-6}$ | – | – |
| Baseline schedule | Cosine | – | – |
| Baseline decay steps | 30,000 | – | – |
| Baseline warm-up | 1,000 | – | – |
| TTT peak learning rate | – | $5 \times 10^{-5}$ | $5 \times 10^{-5}$ |
| Backbone peak learning rate | As above | Frozen | $2.5 \times 10^{-6}$ |
| TTT and backbone post-training schedule | – | – | Constant |
| TTT and backbone post-training warm-up | – | – | 0 |
| Semantic-encoder learning rate | – | – | $5 \times 10^{-5}$ |
| Semantic-encoder warm-up updates | – | – | 0 |
| Semantic-encoder LR schedule | – | – | Constant |
| Optimizer | | AdamW | |
| Adam first-moment decay | | 0.9 | |
| Adam second-moment decay | | 0.95 | |
| Weight decay | $10^{-10}$ | $10^{-5}$ | $10^{-5}$ |
| Gradient clip | 1.0 | 1.0 | 1.0 |
| EMA decay | 0.99 | 0.999 | 0.999 |
| TBPTT segment length | – | 4 blocks | 4 blocks |

### E.3 Real-World Experiments

We train one policy per suite, first fine-tuning $\pi_{0.5}$ for 30,000 updates on the suite's initial expert trajectories. All real-world training uses batch size 128 and action chunks of 20 steps. With the resulting backbone frozen, TTT training uses $T = 1, 8, 32$ for Repeated Picking and Repeated Swinging, and $T = 1, 4, 8, 32, 64$ for Chemistry Experiment, with 10,000 updates per stage. This yields a separate RoboTTT (base) for each suite.

RoboTTT (fair), RoboTTT + DAgger, and SELA-TTT then train for 20,000 updates at the suite's final context length, using the datasets in Appendix D.3. All parameters are unfrozen, following the DAgger masking and parameter-group learning-rate schedules in Appendix E.1. SELA-TTT additionally trains the semantic encoders and uses subtask advancement with FiLM. Table 21 summarizes the suite-specific schedules.

Table 21: Real-world training settings. Post-training budgets are identical for Fair, DAgger, and SELA-TTT.

| Setting | Repeated Picking | Repeated Swinging | Chemistry Experiment |
|---|---|---|---|
| $\pi_{0.5}$ training updates | 30,000 | 30,000 | 30,000 |
| TTT context curriculum | $1, 8, 32$ | $1, 8, 32$ | $1, 4, 8, 32, 64$ |
| TTT updates per stage | 10,000 | 10,000 | 10,000 |
| Total TTT-only updates | 30,000 | 30,000 | 50,000 |
| Post-training context length | 32 | 32 | 64 |
| Post-training updates | 20,000 | 20,000 | 20,000 |
| Global batch size | 128 | 128 | 128 |
| Action horizon | 20 | 20 | 20 |
| TTT peak learning rate | $5 \times 10^{-5}$ | | |
| Post-training backbone peak LR | $2.5 \times 10^{-6}$ | | |
| Initial $\pi_{0.5}$ peak learning rate | $5 \times 10^{-5}$ | | |
| Initial $\pi_{0.5}$ final learning rate | $5 \times 10^{-5}$ | | |
| Initial $\pi_{0.5}$ LR schedule | Linear warm-up, then constant | | |
| Initial $\pi_{0.5}$ warm-up updates | 10,000 | | |
| Post-training TTT and backbone schedule | Constant | | |
| Post-training TTT and backbone warm-up | 0 | | |
| Semantic-encoder learning rate | $5 \times 10^{-5}$ | | |
| Semantic-encoder warm-up updates | 0 | | |
| Semantic-encoder LR schedule | Constant | | |
| Optimizer | AdamW | | |
| Adam first-moment decay | 0.9 | | |
| Adam second-moment decay | 0.95 | | |
| TTT and post-training weight decay | $10^{-5}$ | | |
| Gradient clip | 1.0 | | |
| EMA decay | 0.999 | | |
| TBPTT segment length | 4 blocks | | |

# F Evaluation

## F.1 RoboMME

For the main comparisons, we adopt the official RoboMME evaluation protocol (Dai et al., 2026): 50 episodes per task for each of three checkpoints and three random seeds, totaling 450 rollouts per task, with a maximum of 1,300 environment steps per episode. Success is determined by the benchmark's task-specific checks, and the overall score is the macro-average across tasks. Our RoboTTT and SELA-TTT variants in the main comparison use 450 rollouts per task. Results for external baselines are taken from their reported evaluations. The 50-episode RoboTTT (base) result is retained only where explicitly stated for diagnostic and appendix ablation references. Semantic signals are computed online by the same subtask and intervention-label functions used in Appendix D.1.3, using only the interaction observed so far. RoboMME ablations reported only in the appendix use 50 episodes per task, except the injection-interface and Q-only, KV-only, and QKV read–write comparisons, which use 450 episodes per task. The semantic–context correspondence tests in the main text likewise use 50 continuations per task and label condition (Appendix F.1.1). The separate retrieval and numerical-equivalence diagnostics retain their stated sampling budgets.

### F.1.1 Semantic–Context Correspondence

This evaluation tests the lasting effect of semantic information associated with a fixed interaction history. We use the same subtask-advancement QKV checkpoint without retraining. A fixed rollout-and-correction procedure supplies 50 held-out online rollouts per task, each retained as an uninterrupted Original continuation and selected before comparing conditions. Negative intervals concern an unhighlighted object in PickHighlight, a phase-relative directional deviation in SwingXtimes, or an unrequested container after swapping in ButtonUnmaskSwap. In PickXtimes, the altered positive interval completes an early pick-and-place cycle. Table 1 specifies each label change, with equal-length intervals exchanged in ButtonUnmaskSwap. Other labels are unchanged. Deviations are

corrected before terminal failure, and continuation begins two nonnegative execution chunks after the last altered interval, with repetitions or target selections still remaining.

With $a$ the first affected update and $r$ the continuation boundary, the online rollout caches all layers' fast parameters $\theta_a$, the recorded policy inputs and labels from $a$ through $r-1$, including committed-update noisy actions, and the environment snapshot at $r$. The altered replay holds these inputs fixed and recomputes

$$\theta'_a = \theta_a, \qquad \theta'_{i+1} = \mathcal{F}_i(\theta'_i, \widetilde{\rho}_i), \qquad a \le i < r, \tag{33}$$

where $\widetilde{\rho}_i$ is the replayed label and $\mathcal{F}_i$ updates all fast parameters, including those of $\phi_i$, $W_i$, and $b_i$, with effective features and downstream computations recalculated. Original continues online from $r$ without replay; unmodified replay is used only to validate exact reconstruction. Altered resumes from the saved environment and task state at $r$ with the recomputed fast state, identical current feedback, and the same remaining budget. Subsequent actions and signals follow each condition's own interaction. Each task therefore has 50 online Original continuations and 50 corresponding Altered continuations, isolating the interpretation of a fixed history rather than changing its actions.

**Targeted errors.** The targeted-error rate is the fraction of all 50 continuations that repeat the selected incorrect choice in PickHighlight, exceed the instructed count in PickXtimes, repeat the same phase-relative directional deviation in SwingXtimes, or select the container associated with the negative interval in ButtonUnmaskSwap. The selected object or container is not a later valid target. Each event is counted at most once per continuation using the task predicate or deviation criterion. We report targeted-error rates alongside task success.

### F.1.2 PER-TASK RESULTS FOR ROBOMME

Table 22 reports the external references, and Table 23 reports our RoboTTT baselines and three SELA-TTT signals. Table 24 gives the injection-interface and read–write results underlying Tables 16 and 18. All macro-averages weight the 16 tasks equally.

Table 22: RoboMME per-task success (%): published external references.

| Task | $\pi_{0.5}$ | TTT+ Context | TTT+ Modul | TTT+ Expert | FrameSamp +Modul | MemER | SAM2Act+ | Memory VLA |
|---|---|---|---|---|---|---|---|---|
| Counting | | | | | | | | |
| BinFill | 30.00 | 35.56 | 34.22 | 34.89 | 39.56 | 56.67 | 40.00 | 10.00 |
| StopCube | 6.67 | 3.33 | 2.11 | 4.00 | 42.00 | 0.00 | 0.00 | 0.00 |
| PickXtimes | 42.89 | 62.89 | 65.11 | 63.78 | 87.33 | 79.33 | 76.00 | 17.33 |
| SwingXtimes | 35.56 | 42.44 | 36.67 | 41.33 | 92.00 | 59.33 | 25.33 | 1.33 |
| Permanence | | | | | | | | |
| ButtonUnmask | 22.22 | 22.89 | 22.11 | 22.44 | 25.11 | 72.00 | 32.00 | 8.00 |
| VideoUnmask | 20.44 | 29.78 | 27.22 | 31.78 | 32.67 | 81.33 | 27.33 | 16.00 |
| VideoUnmaskSwap | 18.67 | 18.44 | 25.22 | 19.56 | 24.44 | 38.00 | 18.00 | 4.67 |
| ButtonUnmaskSwap | 6.67 | 14.44 | 14.11 | 18.00 | 18.22 | 21.33 | 26.67 | 4.67 |
| Reference | | | | | | | | |
| PickHighlight | 11.33 | 20.44 | 14.56 | 12.22 | 22.89 | 70.67 | 17.33 | 7.33 |
| VideoRepick | 0.44 | 13.11 | 12.11 | 9.56 | 30.44 | 25.33 | 5.33 | 0.00 |
| VideoPlaceButton | 31.11 | 34.22 | 32.67 | 34.00 | 60.00 | 30.00 | 24.67 | 13.33 |
| VideoPlaceOrder | 25.78 | 20.22 | 22.33 | 22.89 | 32.00 | 26.00 | 20.00 | 4.00 |
| Imitation | | | | | | | | |
| MoveCube | 26.00 | 32.44 | 31.22 | 33.56 | 77.78 | 82.67 | 29.33 | 14.00 |
| InsertPeg | 1.56 | 1.11 | 1.11 | 0.89 | 7.56 | 6.67 | 0.00 | 0.00 |
| PatternLock | 2.89 | 1.56 | 3.56 | 3.11 | 53.56 | 16.67 | 0.00 | 12.00 |
| RouteStick | 4.67 | 3.56 | 7.00 | 5.56 | 66.67 | 12.00 | 0.00 | 1.33 |
| **Macro average** | 17.93 | 22.28 | 21.96 | 22.35 | 44.51 | 42.37 | 21.37 | 7.12 |

Table 23: RoboMME per-task success (%): our methods, with 450 rollouts per task.

| | RoboTTT baselines | | | SELA-TTT: semantic signal | | |
|---|---|---|---|---|---|---|
| Task | Base | Fair | +DAgger | Subtask advancement | Overall goal progress | Atomic-skill optimality |
| Counting | | | | | | |
| BinFill | 32.22 | 43.11 | 56.89 | 73.11 | 79.11 | 57.78 |
| StopCube | 11.33 | 19.78 | 27.11 | 42.00 | 38.44 | 28.89 |
| PickXtimes | 61.11 | 65.56 | 71.11 | 84.89 | 85.56 | 68.44 |
| SwingXtimes | 71.78 | 66.89 | 80.22 | 83.78 | 82.00 | 75.78 |
| Permanence | | | | | | |
| ButtonUnmask | 31.56 | 29.33 | 35.11 | 42.67 | 28.67 | 35.56 |
| VideoUnmask | 26.67 | 36.44 | 40.44 | 50.89 | 45.33 | 38.89 |
| VideoUnmaskSwap | 16.89 | 18.44 | 28.22 | 46.00 | 26.44 | 22.00 |
| ButtonUnmaskSwap | 12.89 | 19.56 | 20.89 | 34.89 | 43.56 | 35.33 |
| Reference | | | | | | |
| PickHighlight | 19.56 | 23.78 | 31.33 | 45.33 | 34.89 | 37.33 |
| VideoRepick | 22.89 | 34.67 | 35.78 | 46.44 | 35.56 | 31.78 |
| VideoPlaceButton | 36.00 | 34.22 | 38.89 | 48.00 | 44.44 | 40.89 |
| VideoPlaceOrder | 32.44 | 38.67 | 39.78 | 48.89 | 39.56 | 44.44 |
| Imitation | | | | | | |
| MoveCube | 59.78 | 66.00 | 75.33 | 88.44 | 88.89 | 77.56 |
| InsertPeg | 5.78 | 21.33 | 40.00 | 57.11 | 54.00 | 39.11 |
| PatternLock | 4.67 | 23.56 | 35.33 | 48.89 | 46.22 | 37.56 |
| RouteStick | 27.11 | 30.44 | 40.44 | 54.00 | 64.67 | 63.56 |
| **Macro average** | 29.54 | 35.74 | 43.56 | 55.96 | 52.33 | 45.93 |

Table 24: RoboMME per-task success (%): injection-interface and read–write variants, with 450 rollouts per task. Adv., Goal, and Skill denote subtask advancement, overall goal progress, and atomic-skill optimality.

| | Input-feature FiLM | | | Raw-QKV FiLM | | | Equivalent-LA read–write | | |
|---|---|---|---|---|---|---|---|---|---|
| Task | Adv. | Goal | Skill | Adv. | Goal | Skill | Q-only | KV-only | QKV |
| Counting | | | | | | | | | |
| BinFill | 46.44 | 44.22 | 35.11 | 62.44 | 72.00 | 50.67 | 66.89 | 63.56 | 73.11 |
| StopCube | 23.78 | 32.67 | 21.11 | 22.22 | 29.78 | 20.67 | 32.44 | 35.78 | 42.00 |
| PickXtimes | 63.78 | 66.89 | 64.22 | 69.56 | 65.78 | 60.00 | 86.44 | 78.00 | 84.89 |
| SwingXtimes | 61.33 | 57.56 | 66.67 | 81.56 | 78.22 | 67.11 | 80.89 | 79.56 | 83.78 |
| Permanence | | | | | | | | | |
| ButtonUnmask | 24.44 | 25.78 | 26.67 | 38.44 | 27.56 | 32.67 | 31.56 | 43.56 | 42.67 |
| VideoUnmask | 25.56 | 29.78 | 22.67 | 37.56 | 40.44 | 40.00 | 33.11 | 36.00 | 50.89 |
| VideoUnmaskSwap | 14.22 | 15.78 | 22.22 | 29.33 | 29.56 | 23.11 | 36.67 | 37.33 | 46.00 |
| ButtonUnmaskSwap | 16.67 | 22.67 | 16.00 | 27.11 | 22.89 | 16.22 | 24.67 | 22.89 | 34.89 |
| Reference | | | | | | | | | |
| PickHighlight | 18.89 | 22.67 | 21.78 | 34.67 | 26.00 | 38.89 | 34.44 | 37.78 | 45.33 |
| VideoRepick | 28.22 | 28.89 | 26.44 | 33.11 | 36.89 | 26.22 | 43.56 | 34.00 | 46.44 |
| VideoPlaceButton | 28.44 | 24.22 | 29.56 | 34.00 | 27.56 | 25.11 | 42.22 | 48.67 | 48.00 |
| VideoPlaceOrder | 30.00 | 33.56 | 35.56 | 36.00 | 34.22 | 32.89 | 52.00 | 43.78 | 48.89 |
| Imitation | | | | | | | | | |
| MoveCube | 61.33 | 59.56 | 64.44 | 67.56 | 70.00 | 65.56 | 80.89 | 78.44 | 88.44 |
| InsertPeg | 13.11 | 18.44 | 15.33 | 44.00 | 38.00 | 37.56 | 45.33 | 32.67 | 57.11 |
| PatternLock | 20.67 | 20.00 | 12.44 | 27.78 | 24.22 | 35.33 | 45.78 | 50.67 | 48.89 |
| RouteStick | 35.11 | 39.33 | 44.67 | 45.11 | 48.00 | 38.22 | 46.00 | 49.56 | 54.00 |
| **Macro average** | 32.00 | 33.88 | 32.81 | 43.15 | 41.94 | 38.14 | 48.93 | 48.26 | 55.96 |

### F.2 RMBENCH

We follow the official RMBench evaluation pipeline (Chen et al., 2026), running 100 episodes for each of the nine tasks with its prescribed reset procedure, task-specific step limit, and success checks. Each task is evaluated with its separately trained policy, and the overall score averages the nine task success rates. Online semantic signals use the same calculation functions as dataset annotation, without access to future interventions or outcomes.

Tables 25 and 26 report per-task success for published external references and our methods, respectively. External results retain their reported evaluation protocols and rounding.

Table 25: RMBench per-task success (%): published external references.

| Task | $\pi_{0.5}$ | X-VLA | Mem-0 | MemoryWAM | EventVLA | DiM-WAM |
|---|---|---|---|---|---|---|
| $M(1)$: single-observation memory | | | | | | |
| Observe and Pick Up | 9 | 9 | 4 | 27 | 21 | 13 |
| Rearrange Blocks | 13 | 13 | 89 | 100 | 96 | 99 |
| Put Back Block | 11 | 18 | 90 | 100 | 95 | 98 |
| Swap Blocks | 24 | 16 | 67 | 100 | 96 | 96 |
| Swap T | 15 | 3 | 14 | 94 | 87 | 97 |
| $M(n)$: multi-observation memory | | | | | | |
| Battery Try | 16 | 26 | 28 | 41 | 35 | 48 |
| Blocks Ranking Try | 6 | 1 | 18 | 100 | 81 | 87 |
| Cover Blocks | 0 | 2 | 68 | 98 | 97 | 56 |
| Press Button | 0 | 0 | 0 | 87 | 3 | 34 |
| **Macro average** | 10.4 | 9.8 | 42.0 | 83.0 | 67.8 | 69.8 |

Table 26: RMBench per-task success (%): our methods, with 100 episodes per task.

| | RoboTTT baselines | | | SELA-TTT: semantic signal | | |
|---|---|---|---|---|---|---|
| Task | Base | Fair | +DAgger | Subtask advancement | Overall goal progress | Atomic-skill optimality |
| $M(1)$: single-observation memory | | | | | | |
| Observe and Pick Up | 17 | 20 | 25 | 32 | 33 | 29 |
| Rearrange Blocks | 87 | 100 | 90 | 95 | 100 | 100 |
| Put Back Block | 68 | 74 | 86 | 100 | 100 | 100 |
| Swap Blocks | 53 | 56 | 78 | 94 | 92 | 93 |
| Swap T | 44 | 57 | 79 | 91 | 86 | 88 |
| $M(n)$: multi-observation memory | | | | | | |
| Battery Try | 34 | 50 | 54 | 71 | 64 | 60 |
| Blocks Ranking Try | 19 | 36 | 41 | 66 | 62 | 64 |
| Cover Blocks | 31 | 53 | 69 | 87 | 89 | 84 |
| Press Button | 34 | 55 | 69 | 92 | 92 | 79 |
| **Macro average** | 43.00 | 55.67 | 65.67 | 80.89 | 79.78 | 77.44 |

### F.3 Real-World Experiments

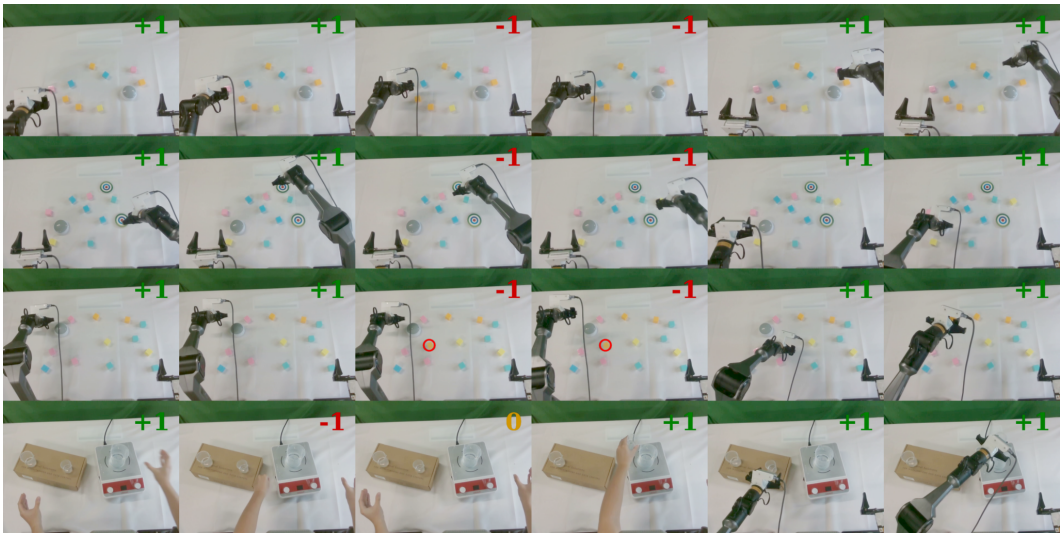



Figure 11: Generalization to unseen switching, continuation, and physical failures, followed by semantic-aware in-context learning (top to bottom).

Methods share task instructions, initial-scene distributions, and execution budgets. The ICL evaluation compares RoboTTT (base), RoboTTT (fair), RoboTTT + DAgger, RoboTTT + DAgger + Int., and SELA-TTT; the failure tests compare the latter three. We evaluate each method for 40 episodes per suite in the main experiments and 20 episodes in each special evaluation, with limits of 4,000 environment steps for Chemistry Experiment and 2,000 for Repeated Picking and Repeated Swinging. A human operator supplies subtask advancement online, starting at 0 and manually switching among $-1$, 0, and $+1$ as execution deviates, is corrected, or advances. The signal affects subsequent decisions only. For RoboTTT + DAgger + Int., the operator intervenes when an error occurs and returns the scene to the state preceding the current subtask while preserving previously completed progress. The intervention observations and executed actions update the TTT fast state before control returns to the policy, without resetting its context or updating slow parameters.

#### F.3.1 SEMANTIC-AWARE IN-CONTEXT LEARNING

We use the Chemistry Experiment suite with 20 prompts paired with previously unseen imperfect human demonstrations carrying subtask-advancement annotations, with one evaluation episode per prompt. Evaluation uses the same demonstration-conditioned instruction format as training. Demonstration construction and annotation follow Appendix D.3, including undesired partial subtask execution followed by rollback. We use each method's task-performance checkpoint and provide the same videos across methods, with semantic annotations available to SELA-TTT.

#### F.3.2 UNSEEN PHYSICAL FAILURES

We use the Repeated Picking suite for the physical-failure test: an operator releases the block after it has been fully lifted but before it is returned to its original position, so that it falls to a location within reach. This interaction is unseen during training. The original instruction and completed progress are retained, while the interrupted cycle remains incomplete. A negative semantic signal marks the drop, after which subtask labels follow the observed recovery progress. Success requires picking up the block again and completing the remaining instructed pick-and-place cycles before pressing the button. Episode counts, baselines, and subsequent intervention rules follow Appendix F.3.

#### F.3.3 UNSEEN SEMANTIC–INTERACTION FAILURES

The switching and continuation tests evaluate unseen correspondences between interaction history and negative semantic signals.

**Switching.**

We use the Repeated Picking suite for the switching test, with approaches to the first selected instruction-compatible block receiving a negative semantic signal for the rest of the episode. Withdrawal and release remain permissible. The policy must switch to another matching block and complete the requested repetitions before pressing the button. The same trigger rule applies to every method, including the intervention baseline, although the initially selected object may differ.

**Continuation.**

We use the Repeated Swinging suite for the continuation test, introducing a negative semantic signal during a designated outward or return movement before all instructed repetitions are complete. The policy must stop swinging, release the block, and press the button rather than continue the remaining repetitions. Triggering uses the same movement phase and criterion across methods, not a fixed elapsed time. The recovery primitives are present in training, but this failure condition is not.

#### F.3.4 EVALUATION METRICS

Task success follows the instructed completion conditions in ordinary evaluation and the task-specific conditions above in the special tests. Timeouts count as failures, and expert-controlled corrections do not count as autonomous progress.

In the Chemistry Experiment, Errors reports the number of episodes that reproduce negatively annotated undesired behavior from the demonstration. Each episode has a binary error indicator: reproducing such behavior sets it to 1 and makes the episode a failure, regardless of subsequent execution; otherwise, it is 0. The reported count sums these indicators over the 20 evaluation episodes.