FAST RESIDUAL MEMORY FOR TEST-TIME TRAINING OF LATENT WORLD MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent world models are typically deployed as static predictors, even though every prediction is eventually confronted with a realized future state that provides direct self-supervised feedback. We study how such feedback can be reused at test time without modifying the pretrained world model itself. We introduce Fast Residual Memory, a trajectory-local fast-weight state that stores recent prediction residuals and retrieves context-dependent corrections for subsequent predictions. Observed residuals are written into the fast state through a lightweight delta-rule update, while learned write and read projections are trained with a support-query objective so that residuals collected from past transitions improve held-out future predictions rather than merely memorizing observed errors. At deployment, the pretrained world model and residual-memory parameters remain fixed, while only the fast state evolves through a causal predict-observe-update process. Across LeWM and DINO-WM, Fast Residual Memory reduces latent prediction error under held-out dynamics shifts, including both interpolation and extrapolation settings. Further analysis shows that the fast state is best understood as a context-dependent, trajectory-local memory of recent model error rather than as a global representation that can be reliably reused across trajectories.

1 INTRODUCTION

World models learn predictive representations of how environments evolve and provide a foundation for planning, control, and embodied decision making (Hafner et al., 2025; Hansen et al., 2024). Recent latent predictive models, including JEPA-style world models, further show that useful environment dynamics can be modeled directly in representation space without reconstructing future observations at the pixel level (Zhou et al., 2025; Maes et al., 2026b). Despite these advances, latent world models are still commonly deployed as fixed predictors. This is particularly restrictive for world models because prediction and observation unfold sequentially: a prediction made at time t is eventually confronted with the realized state at time $t+1$, providing direct self-supervised feedback about where the model was correct and where it was wrong. Standard inference largely discards this information, treating realized prediction errors as evaluation signals rather than experience that could inform subsequent predictions.

This feedback becomes especially valuable when deployment dynamics differ from those seen during training. Changes in physical properties such as mass, friction, gravity, external forces, or action scaling may leave the observation space and task semantics largely unchanged while systematically altering how states evolve. In such cases, a pretrained latent world model can still provide a useful predictive prior, yet exhibit structured residual errors because part of the current transition dynamics is no longer captured by that prior. Once the realized future state becomes available, these residuals provide direct evidence of the local mismatch between the frozen model and the current dynamics. This motivates exploiting residual feedback through a lightweight adaptive state, rather than repeated parameter optimization that can increase test-time cost and modify useful predictive knowledge. The key question is therefore not simply whether prediction errors can be stored, but whether the predictable component of recent errors can be reused to improve future predictions under related contexts.

To exploit this reusable error structure without altering the pretrained predictor, we introduce Fast Residual Memory, a trajectory-local fast-weight state that evolves alongside a frozen latent world

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

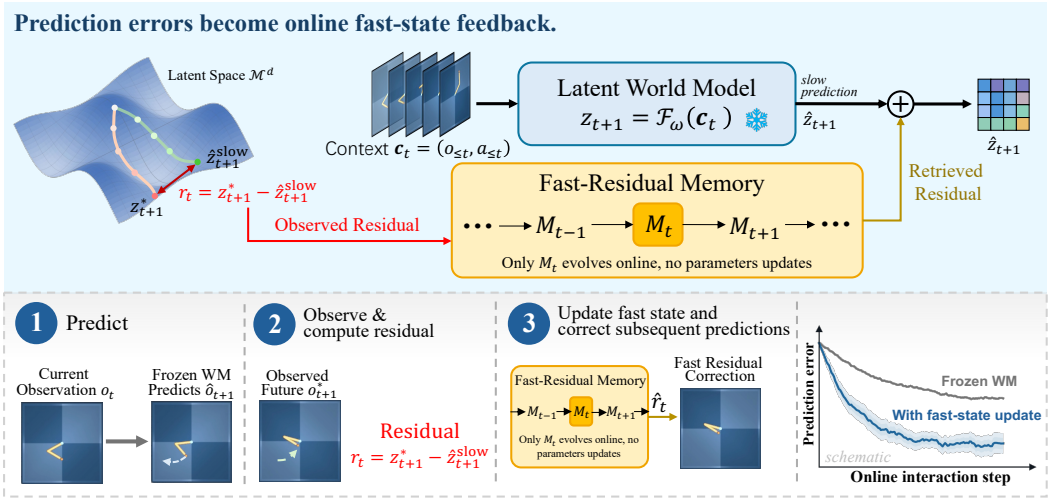


Figure 1: Overview of Fast Residual Memory. Prediction residuals are written into a trajectory-local fast state and retrieved to correct subsequent predictions, while the world model remains frozen.

model. The world model provides a stable predictive prior, while the fast state accumulates how that prior has recently been wrong under encountered predictive contexts. Learned write projections associate each context with its realized residual, and learned read projections retrieve context-relevant corrections for subsequent predictions. Fig. 1 provides an overview of the predict–observe–update process: realized prediction residuals are written into a trajectory-local fast state and retrieved to correct subsequent predictions while the pretrained world model remains fixed. At deployment, the pretrained world model and residual-memory parameters remain fixed; only the trajectory-local fast state evolves through a causal predict–observe–update loop. Test-time adaptation is therefore carried entirely by the evolving fast state, without updating any learned parameters.

Simply retaining past residuals, however, does not guarantee better future predictions. An observed error is useful only to the extent that it captures structure that persists beyond the transition on which it was measured and remains relevant under a new predictive context. We therefore train the memory mechanism with a support-query objective: residuals from a preceding support segment are written into the fast state, while the resulting memory is evaluated only on held-out future transitions. This encourages the learned write and read projections to preserve residual structure that is predictive across transitions, rather than merely reconstructing or memorizing observed errors.

We evaluate Fast Residual Memory on two pretrained latent world models, LeWM and DINO-WM (Maes et al., 2026b; Zhou et al., 2025), under controlled held-out dynamics shifts spanning damping, gravity, wind, density, and action scaling. Across both backbones, the method consistently reduces one-step latent prediction error and generalizes to both interpolation and extrapolation settings. Beyond these gains, our analyses reveal an important property of the learned fast state: its usefulness depends on recent predictive context, can degrade when the underlying dynamics change, and does not transfer reliably across trajectories. These results suggest that prediction residuals are indeed reusable test-time experience, but primarily as a context-dependent, trajectory-local memory of recent model error rather than as a global representation that can be reliably reused across trajectories.

Our contributions are threefold. First, we formulate test-time adaptation for latent world models around the predictive feedback naturally revealed by interaction, treating realized prediction residuals as reusable experience rather than discarded evaluation signals. Second, we introduce Fast Residual Memory, a trajectory-local fast-weight state that associates predictive contexts with the errors of a frozen world model; its learned write and read projections are optimized with a support-query objective so that stored residuals improve held-out future predictions rather than merely reconstructing observed errors. Third, across LeWM and DINO-WM under held-out dynamics shifts, we show that residual feedback consistently improves online latent prediction and generalizes across interpolation and extrapolation settings, while further revealing that its usefulness is inherently context-dependent and trajectory-local.

2 Related Work

Latent predictive world models. Learning environment dynamics in compact latent spaces has long been central to visual world modeling. Early methods such as World Models and PlaNet learn compressed state representations and latent transition models for control and planning (Ha & Schmidhuber, 2018; Hafner et al., 2019), while the Dreamer and TD-MPC families further demonstrate scalable decision making through imagined rollouts and latent-space trajectory optimization Hafner et al. (2025); Hansen et al. (2024). More recently, joint-embedding predictive architectures shift the focus from reconstructing observations to predicting informative representations directly: I-JEPA and V-JEPA establish feature prediction as a self-supervised learning principle, while DINO-WM, V-JEPA 2-AC, LeWorldModel, and C-JEPA extend joint-embedding prediction toward action-conditioned, object-centric, and physically grounded world modeling (Assran et al., 2023; Bardes et al., 2024; Zhou et al., 2025; Assran et al., 2025; Maes et al., 2026b; Nam et al., 2026; Terver et al., 2026).

Recent work has also explored adapting world models during deployment. AdaJEPA updates latent dynamics from newly observed transitions, while SPREAD and WorldAgen perform lightweight online adaptation of model components (Wang et al., 2026; Moon & Kim, 2026; Wan et al., 2026). WAM-TTT similarly keeps a pretrained world-action model frozen while adapting a lightweight memory from human videos through self-supervised prediction (Feng et al., 2026). In contrast, we use prediction residuals revealed by the model's own interaction trajectory to update a trajectory-local fast state for correcting subsequent latent predictions, while keeping all persistent model and adapter parameters fixed at deployment.

Test-time training and adaptation. Test-time learning enables pretrained models to improve during deployment using information from the test stream. Early Test-Time Training (TTT) turns unlabeled test samples into auxiliary self-supervised learning problems, while methods such as Tent, CoTTA, EATA, and SAR extend this paradigm toward entropy-based, continual, and more stable adaptation under distribution shift (Sun et al., 2020; Wang et al., 2021; 2022; Niu et al., 2022; 2023). More recent work further considers practical deployment under evolving test streams: RoTTA targets temporally correlated and dynamically changing test distributions (Yuan et al., 2023), while EcoTTA studies memory-efficient continual adaptation with a frozen backbone and lightweight adaptive modules (Song et al., 2023). Zhang et al. (2026) further study how online optimization can be made more effective and efficient during inference. Unlike these settings, world models receive direct predictive feedback because the outcome of a prediction is naturally revealed by subsequent interaction. We use this realized feedback to update a trajectory-local fast state, rather than the world-model parameters, for correcting future predictions.

Fast weights and learning at test time. Fast-weight models maintain rapidly changing states as temporary memory alongside shared model parameters. Early work used fast weights to store information from the recent past at a faster adaptation timescale than conventional model parameters (Ba et al., 2016). Fast Weight Programmers later interpreted matrix-valued hidden states as associative key-value memories with delta-rule updates (Schlag et al., 2021). More recent approaches view evolving hidden states as online learners or trainable memories: Longhorn derives state transitions from an online learning objective (Liu et al., 2025), while TTT layers and Titans learn hidden states or neural memories that are updated during inference (Sun et al., 2025; Behrouz et al., 2026). RoboTTT further applies test-time-updated fast weights to robot policies, compressing long visuomotor histories for contextual conditioning (Jiang et al., 2026). These methods primarily use fast-changing states to store or transform information from preceding context. In contrast, Fast Residual Memory stores context-dependent prediction errors from a frozen latent world model and retrieves them as corrections for subsequent predictions, while the pretrained world model and residual-memory parameters remain fixed at deployment.

3 Method

We now formalize the fast residual memory introduced above. Fig. 2 summarizes the complete Fast Residual Memory mechanism, including residual writing and retrieval, support–query training, and causal test-time updates. Our method augments a frozen pretrained latent world model with a trajectory-local fast state that stores prediction residuals observed during interaction. We first

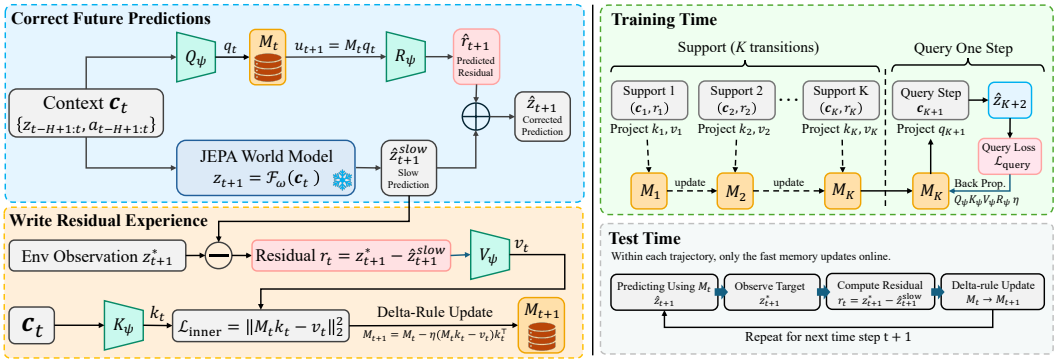


Figure 2: Fast Residual Memory. The frozen world model provides a slow prediction, while the fast memory retrieves a context-dependent residual correction. Revealed prediction residuals are written into the trajectory-local memory through a delta-rule update. The memory mechanism is trained with a support-query objective, and at test time only the fast memory evolves online.

define the predictive feedback available from each transition, and then describe how this residual information is written to and retrieved from the fast-weight memory for subsequent prediction.

3.1 Problem Setup and Predictive Feedback

Let o_t and a_t denote the observation and action at time t. A pretrained encoder maps each observation to a latent representation,

$$z_t^* = E(o_t). \tag{1}$$

Given a recent context c_t, which contains a history of latent states and actions depending on the underlying world model, a frozen predictor F_ω produces the corresponding slow prediction,

$$\hat{z}^{\,s}_{t+1} = F_\omega(c_t), \tag{2}$$

where the superscript s denotes the output of the slow pretrained world model. The parameters of the pretrained world model remain fixed throughout both training of the fast-memory mechanism and test-time deployment. Once the future observation o_{t+1} becomes available, its latent representation provides a target for evaluating the preceding prediction. We define the resulting prediction residual as

$$r_t = z^*_{t+1} - \hat{z}^{\,s}_{t+1}. \tag{3}$$

This residual isolates the part of the realized transition that is not explained by the frozen predictor. Importantly, r_t becomes available only after the prediction at time t has been made, which gives a causal stream of predictive feedback during interaction. Our goal is to use residuals from previously observed transitions to improve predictions on subsequent ones, without modifying the pretrained world model itself.

3.2 Fast-Weight Residual Memory

Given the residual signal introduced above, we seek a lightweight mechanism that can accumulate information from previously observed prediction errors and reuse it to correct future predictions. This is useful when residual information remains locally predictive across nearby transitions, while its relevance to a new prediction depends on the current predictive context. To this end, we represent the test-time state as a fast-weight matrix $M_t \in \mathbb{R}^{d_v \times d_k}$, initialized to zero at the beginning of each trajectory.

M_t is a trajectory-local state rather than a learned model parameter, and evolves online as new transitions are observed. It is accessed through learned write and read projections, which determine how residual experience is encoded into memory and how relevant corrections are retrieved for future contexts.

Writing residual experience. Once the transition at time t has been observed, the predictive context c_t and its corresponding residual r_t are mapped into a key-value pair,

$$k_t = \text{norm}\left(K_\psi(c_t)\right), \qquad v_t = V_\psi(r_t), \tag{4}$$

where K_ψ and V_ψ are learned projections. The key specifies the predictive context under which the slow world model made an error, while the value encodes that error in a compact residual space. The pair (k_t, v_t) therefore represents an association between a predictive context and the corresponding error of the frozen world model. Importantly, the fast state does not explicitly store past observations or latent states. Instead, it progressively compresses how the slow predictor has been wrong under previously encountered contexts.

We write this residual association into the current fast state using a delta-rule update(Schlag et al., 2021),

$$M_{t+1} = M_t - \eta(M_t k_t - v_t)k_t^\top, \tag{5}$$

where η denotes the fast update rate. This update is equivalent to one gradient-descent step on the local associative objective

$$\ell_t^{\text{inner}} = \frac{1}{2}\|M_t k_t - v_t\|_2^2. \tag{6}$$

In particular, its gradient with respect to the fast state is

$$\nabla_{M_t} \ell_t^{\text{inner}} = (M_t k_t - v_t)k_t^\top, \tag{7}$$

so the delta-rule update in Eq. 5 follows directly from a gradient step on M_t.

Thus, each observed transition modifies the fast state according to the residual information that is not already captured by the current memory. As additional transitions are observed, their residual structure is continuously compressed into the fixed-size state M_t.

Retrieving residual corrections. Because residuals are generally context dependent, not every stored error should contribute equally to every future prediction. The query projection therefore determines which residual associations in the fast state are relevant to the current predictive context. For a new predictive context c_t, we construct a normalized query and retrieve the corresponding residual correction as

$$q_t = \text{norm}\left(Q_\psi(c_t)\right), \qquad \hat{r}_t = R_\psi\left(M_t q_t\right), \tag{8}$$

where Q_ψ is a learned query projection and R_ψ maps the retrieved fast-state representation back into the latent prediction space. The final test-time prediction combines the frozen world-model prediction with the retrieved residual correction,

$$\hat{z}_{t+1} = F_\omega(c_t) + \hat{r}_t. \tag{9}$$

This construction gives the fast state a simple write-read interpretation. Observed prediction errors are written into M_t through (K_ψ, V_ψ), while future contexts retrieve relevant corrections through (Q_ψ, R_ψ). The pretrained predictor therefore continues to provide the primary latent prediction, and the fast-weight memory contributes only the component that can be inferred from residual experience accumulated earlier in the trajectory.

3.3 Training the Residual Memory

Eq. 5 specifies how residual information is accumulated within a trajectory. We now describe how the shared memory parameters are trained so that residual experience collected from past transitions becomes predictive of held-out future errors, rather than merely reconstructing the transitions on which it was written.

For each training sequence, we divide a contiguous set of transitions into a support segment $\mathcal{S}$ followed by a query segment $\mathcal{Q}$. Starting from an empty fast state, we process the support transitions sequentially using Eq. 5 in Section 3.2, producing a support-conditioned memory $M_\mathcal{S}$. The resulting memory therefore summarizes residual experience from the support sequence before any query target is observed.

We then keep $M_\mathcal{S}$ fixed and evaluate the model on the subsequent query transitions. The shared memory parameters are optimized according to the held-out prediction objective

$$\mathcal{L}_{\text{train}} = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}} D\left(\hat{z}_{q+1}(M_\mathcal{S}), z_{q+1}^*\right), \tag{10}$$

where $\hat{z}_{q+1}(M_\mathcal{S})$ denotes the residual-corrected prediction defined in Section 3.2, and D is the latent-space prediction loss.

Crucially, query transitions are not written into the fast state when computing this objective. The model must therefore use residual information acquired from the preceding support sequence to improve prediction on held-out future transitions, rather than directly fitting the query targets. This objective therefore favors residual representations that remain predictive beyond the transitions on which they were written. In this way, the query loss shapes the shared write and read representations according to their usefulness for subsequent prediction.

We train this objective using trajectories collected under diverse dynamics conditions, while keeping the pretrained world model frozen. Only the shared residual-memory parameters are optimized during this stage. At deployment, these parameters are fixed and the fast state alone evolves online as new transitions are observed.

3.4 Test-Time Training Protocol

At deployment, the pretrained world-model parameters ω and the learned residual-memory parameters ψ are fixed. The fast state is initialized as $M_0 = 0$ at the beginning of each test trajectory and is the only component that evolves online.

For each transition, the model first predicts the next latent state using the current fast state through Eq. 9. After the corresponding observation becomes available, the realized target z^*_{t+1} determines the residual in Eq. 3, which is then written into the fast state using the update in Eq. 5. The deployment loop is therefore

$$\boxed{M_t \xrightarrow{\text{predict}} \hat{z}_{t+1} \xrightarrow{\text{observe}} z^*_{t+1} \xrightarrow{\text{update}} M_{t+1}.}$$

This ordering ensures that the residual associated with the current transition cannot influence its own prediction and is available only for subsequent predictions. Unlike training, where support transitions are used to construct a fixed memory before evaluation on held-out queries, test-time prediction and memory updates are interleaved continuously along the trajectory.

No learned parameters are updated at deployment; test-time adaptation is carried entirely by the delta-rule evolution of M_t. In all experiments, the fast state is reset at the start of each trajectory.

4 Experiments

4.1 Experimental Setup

We evaluate Fast Residual Memory on two pretrained latent world models, LeWM (Maes et al., 2026b) and DINO-WM (Zhou et al., 2025), using their publicly released checkpoints. The LeWM experiments build on the `stable-worldmodel` framework (Maes et al., 2026a) for environment construction and evaluation under controllable dynamics variations. LeWM serves as the primary evaluation platform, with experiments on PushT, TwoRooms, and Reacher. We introduce controlled dynamics shifts while preserving the task structure and visual semantics. In PushT, we vary the block damping coefficient and the magnitude of a fixed horizontal gravity in the environment. In TwoRooms, we vary the wind speed that perturbs the agent motion. In Reacher, we vary the density of the arm and finger links, thereby changing their response to the same joint torques. We additionally evaluate the same method on DINO-WM using PushT, Wall, and PointMaze as a cross-backbone validation.

For each environment, the residual-memory mechanism is trained across a subset of parameterized dynamics values while the pretrained world model remains frozen. Evaluation is performed on held-out dynamics conditions, including both interpolation within the training range and extrapolation beyond it. For each held-out dynamics value, we evaluate on 50 test trajectories, and aggregate results uniformly across the evaluated test conditions. At test time, the shared residual-memory parameters are fixed, the fast-weight state is initialized to zero at the beginning of each trajectory, and prediction follows the causal predict–observe–update protocol described in Section 3.

We compare three prediction modes throughout the experiments: the frozen pretrained world model (**Base**), the learned residual pathway without trajectory-specific fast-weight information (**Adapter-0**), and the complete fast-weight test-time training model (**TTT**). We primarily report one-step latent

Table 1: Main results under held-out dynamics shifts. Lower prediction loss is better; higher relative gain is better.

Backbone	Method	PushT–Damping		PushT–Gravity		TwoRooms–Wind		Reacher–Density	
		Loss ↓	Rel. gain ↑	Loss ↓	Rel. gain ↑	Loss ↓	Rel. gain ↑	Loss ↓	Rel. gain ↑
LeWM	Base	0.101	–	0.041	–	0.350	–	0.082	–
	Adapter-0	0.102	–	0.043	–	0.333	–	0.082	–
	TTT	**0.093**	8.71%	**0.033**	20.63%	**0.271**	31.25%	**0.053**	36.45%
Backbone	**Method**	**PushT–Damping**		**PushT–Gravity**		**Wall–Wind**		**PointMaze–Action Gain**	
		Loss ↓	Rel. gain ↑	Loss ↓	Rel. gain ↑	Loss ↓	Rel. gain ↑	Loss ↓	Rel. gain ↑
DINO-WM	Base	0.217	–	0.153	–	0.229	–	0.027	–
	Adapter-0	0.216	–	0.153	–	0.227	–	0.027	–
	TTT	**0.187**	**12.69%**	**0.132**	**13.45%**	**0.211**	**7.25%**	**0.022**	**17.97%**

prediction error and relative gain over the frozen baseline

$$G_{\mathrm{rel}} = \frac{L_{\mathrm{Base}} - L_{\mathrm{TTT}}}{L_{\mathrm{Base}} + \epsilon}. \tag{11}$$

Detailed environment interventions, dynamics values, and visualizations under identical action sequences are provided in Appendix A.

4.2 Test-Time Training under Dynamics Shifts

We first evaluate whether Fast Residual Memory improves prediction under held-out dynamics shifts. Table 1 reports the main results for both LeWM and DINO-WM across the considered environments and dynamics interventions.

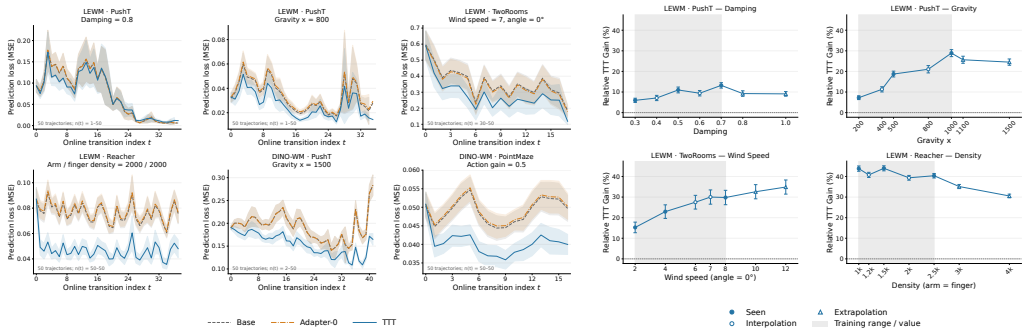
Across LeWM environments, TTT consistently reduces one-step latent prediction error relative to the frozen world model, with Adapter-0 providing a control for improvements that arise from the residual pathway alone. The additional gain of TTT over Adapter-0 therefore reflects the contribution of trajectory-specific fast-weight updates. We observe this behavior across different dynamics shifts, including changes in damping, external forces, wind disturbance, and link density.

We further apply the same Fast Residual Memory formulation to DINO-WM on PushT, Wall, and PointMaze. The resulting improvements show that the benefit of residual test-time training is not specific to a particular latent world-model architecture. Together, these results establish that prediction residuals collected online can provide useful corrections under unseen dynamics while leaving the pretrained world model itself unchanged. Additional comparisons with simple residual-reuse strategies are provided in Appendix E, examining whether the observed gains can be explained by directly reusing previously observed prediction errors.

4.3 Online Test-Time Learning and Dynamics Generalization

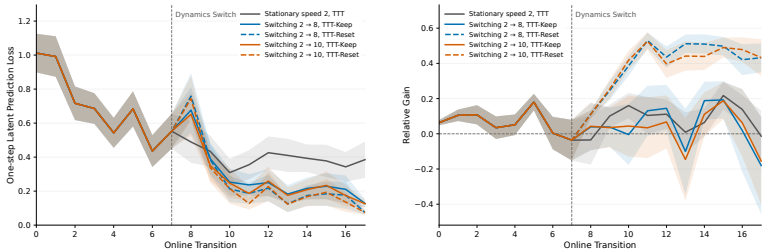
We next examine how the benefit of Fast Residual Memory evolves during online interaction. Fig. 3 reports one-step prediction loss as a function of the online transition index for representative held-out dynamics conditions. The Base and Adapter-0 predictors remain fixed throughout each trajectory, whereas TTT continuously updates its trajectory-local fast state after each realized transition. Since the fast state is initialized from zero, differences that emerge over time reflect information accumulated from previously observed prediction residuals.

Across environments, we compare the evolution of Base, Adapter-0, and TTT prediction losses as additional transitions become available. This analysis complements the trajectory-level results in Table 1 by showing whether the improvement of TTT is acquired online rather than arising solely from the learned residual pathway. In particular, a growing separation between TTT and Adapter-0 indicates that newly observed residuals provide useful information for correcting subsequent predictions.



(a) Online Learning (b) Dynamics Generalization

Figure 3: Online learning and dynamics generalization. (a) TTT maintains lower online prediction loss than Base and Adapter-0 as residual feedback is accumulated. (b) Relative TTT gain remains positive across seen, interpolation, and extrapolation dynamics, indicating generalization of the learned residual-memory mechanism.



(a) Residual Persistence (b) Relative Gain over Base

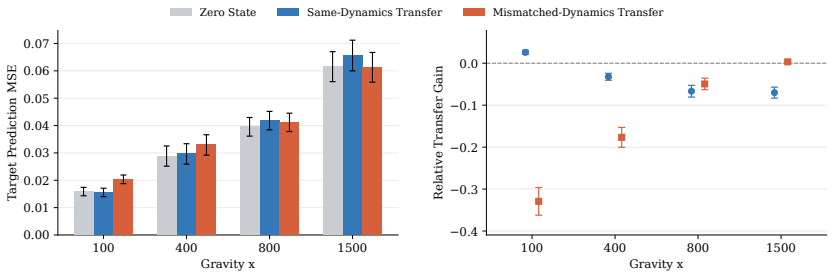
Figure 4: Residual persistence under dynamics changes. (a) Online prediction loss and (b) relative gain over the frozen world model in TwoRooms, comparing stationary wind with speed switches from 2 to 8 and 2 to 10. Keep retains the accumulated fast state, while Reset clears it at the dynamics switch (vertical dashed line). Curves show mean ± SEM across 50 trajectories.

We further evaluate the same trained residual-memory model on held-out dynamics values, including both interpolation within the training range and extrapolation beyond it. For these conditions, the pretrained world model and residual-memory parameters remain fixed; only the fast state evolves online. This evaluation tests whether the learned write and retrieval mechanism can exploit predictive feedback under dynamics that were not encountered during training. Complete per-condition online adaptation curves and the full DINO-WM dynamics sweep are provided in Appendix D.

4.4 Prediction Loss under Dynamics Changes

We investigate how Fast Residual Memory behaves when the underlying dynamics change during deployment. Using LeWM on TwoRooms, we fix the wind direction at 0° and introduce mid-trajectory wind-speed switches from 2 to 8 and from 2 to 10, representing changes to a training dynamics value and an unseen extrapolation value, respectively. We compare two strategies: **Keep**, which retains the accumulated fast state, and **Reset**, which clears it at the switch before resuming online updates. A stationary setting with wind speed 2 serves as a reference.

Fig. 4 shows the online prediction loss and relative gain over the frozen world model. Both switching conditions exhibit substantial prediction gains after the dynamics change, demonstrating that residual test-time training remains beneficial under non-stationary dynamics. Notably, Reset generally achieves larger gains than Keep after the switch, suggesting that previously accumulated residual information can become less relevant when the underlying dynamics change.



(a) Prediction Error

(b) Relative Transfer Gain

Figure 5: Cross-trajectory fast-state transfer under gravity shifts. (a) Prediction error for zero-initialized, same-dynamics, and mismatched-dynamics fast states. (b) Relative transfer gain over the zero-initialized state. Mismatched results are averaged across all other evaluated dynamics conditions. Error bars indicate SEM across target trajectories.

These results highlight both the benefit and the limitation of residual persistence: fast-weight memory can continuously incorporate new predictive feedback, but retaining outdated residual information is not always advantageous when environment dynamics change.

4.5 Fast-State Transferability

We further investigate whether the fast state retains residual information that can be reused across trajectories. For each dynamics condition, we construct fast states from independent source trajectories through sequential online updates. We then freeze these states and evaluate them on disjoint target trajectories without further adaptation. We compare three settings: a zero-initialized state (**Zero**), states transferred from trajectories with the same dynamics (**Same**), and states transferred from trajectories with different dynamics (**Mismatched**). For the mismatched setting, we average prediction losses across source states from all other evaluated dynamics conditions.

Fig. 5 presents the transfer results on LeWM PushT under gravity shifts. At lower gravity values, same-dynamics transfer performs better than mismatched transfer, with a small positive gain at the lowest gravity setting. However, same-dynamics transfer does not consistently outperform the zero-initialized state. At higher gravity values, mismatched transfer can achieve comparable or lower prediction error than same-dynamics transfer.

These results indicate that the usefulness of transferred fast states cannot be determined by dynamics compatibility alone. Since residual corrections are retrieved through context-dependent queries, differences in trajectory contexts may also affect transferability. This observation highlights the trajectory-local nature of Fast Residual Memory and motivates resetting the fast state at the beginning of each trajectory rather than directly reusing states accumulated elsewhere.

5 Conclusion

We introduced Fast Residual Memory, a test-time training framework that turns realized prediction errors into reusable online experience for latent world models. The pretrained world model and residual-memory parameters remain fixed at deployment, while a trajectory-local fast state accumulates recent residual information and retrieves context-dependent corrections for subsequent predictions. Across LeWM and DINO-WM, this mechanism improves prediction under held-out dynamics shifts, including both interpolation and extrapolation settings. Our analyses further show that the usefulness of the fast state depends on recent predictive context: previously accumulated residual information can become less relevant when the dynamics change, and fast states do not transfer reliably across trajectories. These findings suggest that prediction residuals are most useful as local, continually updated experience rather than as a global representation intended for reuse across trajectories. Future work could explore mechanisms for detecting when stored residual information has become stale and for selectively retaining, resetting, or transferring fast-state information across changing dynamics and trajectories.

AI USE STATEMENT

Generative AI tools were used in a limited supporting role during the preparation of this manuscript. Their use was primarily focused on improving the clarity, readability, and presentation of author-written material. This included revising wording for concision and fluency, improving the formatting and presentation of tables and text, checking and standardizing reference formatting, and identifying minor inconsistencies in notation, terminology, and cross-references across different sections of the manuscript.

We also used generative AI tools for limited code-maintenance tasks, such as small-scale refactoring, cleanup, and improving code readability. These uses did not replace the authors' implementation or verification of the proposed method. AI-assisted suggestions were treated as editorial or engineering aids rather than as authoritative outputs.

All AI-assisted text, formatting changes, reference edits, and code modifications were reviewed by the authors before being incorporated. The authors independently verified the technical content, experimental procedures, reported results, and final presentation, and take full responsibility for the content of the paper.

ETHICS STATEMENT

This work studies test-time adaptation for latent world models using simulated control environments and publicly released pretrained models. It does not involve human subjects, personal data, or sensitive user information. The proposed method is intended to improve the robustness of predictive models under changing dynamics; as with other world-modeling techniques, downstream applications in embodied or autonomous systems should consider the reliability and safety implications of prediction errors under distribution shift.

REPRODUCIBILITY STATEMENT

We describe the Fast Residual Memory formulation, training objective, and causal test-time update procedure in Section 3, and report the experimental protocol, evaluation metrics, and dynamics-shift settings in Section 4. Our experiments use publicly released checkpoints for LeWM and DINO-WM. Detailed dynamics values, model and optimization configurations, and additional implementation details are provided in the appendix and supplementary material to support reproduction of the reported results. All experimental results are reported faithfully based on the corresponding runs and evaluation protocols described in the paper.

REFERENCES

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15619–15629, 2023. doi: 10.1109/CVPR52729.2023.01499.

Mahmoud Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, Sergio Arnaud, Abha Gejji, Ada Martin, Francois Robert Hogan, Daniel Dugas, Piotr Bojanowski, Vasil Khalidov, Patrick Labatut, Francisco Massa, Marc Szafraniec, Kapil Krishnakumar, Yong Li, Xiaodong Ma, Sarath Chandar, Franziska Meier, Yann LeCun, Michael Rabbat, and Nicolas Ballas. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

Jimmy Ba, Geoffrey E. Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, volume 29, pp. 4331–4339, 2016.

Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mido Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video. *Transactions on Machine Learning Research*, 2024.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *Advances in Neural Information Processing Systems*, 38:113506–113543, 2026.

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, 2026.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pp. 2451–2463. Curran Associates, Inc., 2018. URL https://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution. https://worldmodels.github.io.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565, 2019.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse control tasks through world models. *Nature*, 640(8059):647–653, 2025. doi: 10.1038/s41586-025-08744-2.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2024.

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. *arXiv preprint arXiv:2607.15275*, 2026.

Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and Qiang Liu. Longhorn: State space models are amortized online learners. In *International Conference on Learning Representations*, 2025.

Lucas Maes, Quentin Le Lidec, Luiz Facury, Nassim Massaudi, Ayush Chaurasia, Francesco Capuano, Richard Gao, Taj Gillin, Dan Haramati, Damien Scieur, Yann LeCun, and Randall Balestriero. stable-worldmodel: A platform for reproducible world modeling research and evaluation, 2026a. URL https://arxiv.org/abs/2605.21800.

Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. Leworldmodel: Stable end-to-end joint-embedding predictive architecture from pixels. *arXiv preprint arXiv:2603.19312*, 2026b.

Jihun Moon and {Seong Woo} Kim. Spread: Scalable pre-trained world model for adaptive dynamics model. *IEEE Robotics and Automation Letters*, 11(6):7460–7467, June 2026. ISSN 2377-3766. doi: 10.1109/LRA.2026.3688061. Publisher Copyright: © 2016 IEEE.

Heejeong Nam, Quentin Le Lidec, Lucas Maes, Yann LeCun, and Randall Balestriero. Causal-jepa: Learning world models through object-level latent interventions. *arXiv preprint arXiv:2602.11389*, 2026.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *The Internetional Conference on Machine Learning*, 2022.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Zhiquan Wen, Yaofo Chen, Peilin Zhao, and Mingkui Tan. Towards Stable Test-Time Adaptation in Dynamic Wild World. In *International Conference on Learning Representations*, 2023. URL https://mlanthology.org/iclr/2023/niu2023iclr-stable/.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pp. 9355–9366. PMLR, 2021.

Junha Song, Jungsoo Lee, In So Kweon, and Sungha Choi. Ecotta: Memory-efficient continual test-time adaptation via self-distilled regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11920–11929, June 2023.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pp. 9229–9248. PMLR, 2020.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (Learn at test time): RNNs with expressive hidden states. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 57503–57522. PMLR, 2025.

Basile Terver, Tsung-Yen Yang, Jean Ponce, Adrien Bardes, and Yann LeCun. What drives success in physical planning with joint-embedding predictive world models? *Transactions on Machine Learning Research*, 2026.

Chi Wan, Kangrui Wang, Yuan Si, Pingyue Zhang, and Manling Li. Worldagen: Unified state-action prediction with test-time world model training. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 18584–18592, 2026.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully Test-Time Adaptation by Entropy Minimization. In *International Conference on Learning Representations*, 2021. URL `https://mlanthology.org/iclr/2021/wang2021iclr-tent/`.

Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. Continual test-time domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7201–7211, June 2022.

Ying Wang, Oumayma Bounou, Yann LeCun, and Mengye Ren. Adajepa: An adaptive latent world model. *arXiv preprint arXiv:2606.32026*, 2026.

Longhui Yuan, Binhui Xie, and Shuang Li. Robust test-time adaptation in dynamic scenarios. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15922–15932, June 2023.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William Freeman, and Hao Tan. Test-time training done right. In *International Conference on Learning Representations*, volume 2026, pp. 157604–157638, 2026.

Gaoyue Zhou, Hengkai Pan, Yann Lecun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 79115–79135. PMLR, 2025.

A Experimental Details

A.1 Environment Dynamics Interventions

Our controlled dynamics interventions alter the environment transition dynamics while preserving the action and task semantics. The interventions affect how the same action is translated into the next state, without changing the observation space or task definition.

PushT. We consider two dynamics interventions: damping and horizontal gravity. The agent is a kinematic disk controlled toward an action-specified target position by a PD controller, while the T-shaped object is a dynamic rigid body that moves only through contact forces, gravity, and damping. Changing the damping coefficient modifies how quickly the T-shaped object loses its linear and angular velocity after being pushed: lower damping causes faster decay, while larger values preserve motion for longer. For the gravity intervention, we vary only the horizontal component g_x. This acceleration acts on both the controlled disk and the dynamic object, causing the same action sequence to induce different horizontal motion while preserving the underlying task.

TwoRooms. The agent position is updated directly at every environment step according to the action-induced displacement plus a constant wind displacement,

$$\Delta p = a\, v_{\text{agent}} + w(\cos\theta, \sin\theta), \tag{12}$$

where w denotes the wind speed. We fix the wind direction to 0°, corresponding to the positive horizontal direction, and vary only its magnitude. Thus, even under the same action sequence, larger wind speeds systematically shift the agent farther in the horizontal direction. Collision handling remains unchanged, so neither the action nor the wind can move the agent through walls. The DINO-WM Wall environment uses the same wind intervention as TwoRooms: the wind direction is fixed while its magnitude is varied to introduce a constant additive displacement under the same action commands.

Reacher. The two-dimensional action controls the shoulder and wrist joint motors through bounded torques. We vary the density of the arm and finger geometries while leaving the applied torques and joint damping unchanged. Since body mass and rotational inertia scale with density, increasing density produces smaller angular acceleration under the same motor command and changes how the joints respond over time. The resulting fingertip trajectory therefore differs even when the action sequence is identical. Gravity acts along the out-of-plane axis and does not directly generate torque around the planar joints.

PointMaze. We vary the action gain g, which scales the motor strength while leaving the maze layout, object properties, damping, and gravity unchanged. After clipping each action component to $[-1, 1]$, the actuator gear is set to $100g$, so the resulting force is

$$F = 100\, g\, a.$$

The same action command therefore produces different accelerations and velocities as g changes, which in turn alters the subsequent position trajectory through the simulator dynamics. Larger gains produce stronger motor forces and faster motion, while smaller gains produce weaker responses to the same command.

A.2 Dynamics Values and Evaluation Splits

We evaluate Fast Residual Memory on two pretrained latent world-model backbones, LeWM and DINO-WM, using their publicly released checkpoints. Our experiments introduce controlled changes to environment dynamics while preserving the observation space and task semantics. For each environment and shift type, the residual-memory mechanism is trained on trajectories collected under a discrete set of dynamics values, while the pretrained world model remains frozen.

We evaluate three types of dynamics conditions. **Seen** conditions use dynamics values that are included during residual-memory training. **Interpolation** conditions use held-out values that lie within the range spanned by the training dynamics, whereas **extrapolation** conditions use held-out values outside that range. Importantly, interpolation and extrapolation conditions are not used to train the residual-memory parameters; at test time, only the trajectory-local fast state evolves from prediction residuals observed online.

For LeWM, we evaluate PushT under changes in block damping and horizontal gravity, TwoRooms under changes in wind speed, and Reacher under changes in the densities of the arm and finger links. For DINO-WM, we consider PushT under damping and horizontal-gravity shifts, Wall under wind-speed shifts, and PointMaze under changes in action gain. The exact dynamics values used for training and evaluation are summarized in Table A.1. Unless stated otherwise, each evaluation dynamics value is tested on 50 trajectories. Results aggregated across multiple dynamics conditions assign equal weight to each evaluated condition.

Table A.1: Dynamics values used for training and evaluation. Seen values coincide with the dynamics used to train the residual-memory mechanism. Interpolation and extrapolation values are held out during training.

Backbone	**Environment**	**Shifted parameter**	**Training / seen**	**Interpolation**	**Extrapolation**
LeWM	PushT	Damping	0.3, 0.5, 0.7	0.4, 0.6	0.8, 1.0
LeWM	PushT	Gravity	200, 500, 1000	400, 800	1100, 1500
LeWM	TwoRooms	Wind speed	2, 4, 8	6, 7	10, 12
LeWM	Reacher	Arm/finger density	1000, 1500, 2500	1200, 2000	3000, 4000
DINO-WM	PushT	Damping	0.3, 0.5, 0.7	0.4, 0.6	0.8, 1.0
DINO-WM	PushT	Gravity	200, 500, 1000	400, 800	1100, 1500
DINO-WM	Wall	Wind speed	2, 4, 8	6, 7	10, 12
DINO-WM	PointMaze	Action gain	0.75, 1.0, 1.25	0.875, 1.125	0.5, 1.5

A.3 Visualizing Dynamics Shifts under Identical Actions

To illustrate the effect of the controlled dynamics interventions, we replay the same action sequence from the same initial state under the original and modified environments. As shown in Fig. 6, the observations are initially aligned but gradually diverge as the altered dynamics accumulate over time. This behavior is consistent across the considered interventions, including PushT damping and gravity, PointMaze action gain, TwoRooms wind speed, and Reacher density. The comparison illustrates that the experiments preserve the task and action semantics while changing how the environment evolves in response to the same controls.

B Implementation Details of Fast Residual Memory

For LeWM, we keep the pretrained encoder, action encoder, and predictor frozen throughout both residual-memory training and test-time evaluation. The memory context is constructed by concatenating the latent-state and action embeddings over the predictor history window and flattening them into a single vector. If the predictor uses H history steps, the resulting context dimension is $H(d_z + d_a)$, where d_z and d_a denote the latent and action-embedding dimensions, respectively.

The residual-memory module consists of four learned linear projections: separate query and key projections from the context to a 128-dimensional key space, a value projection from the prediction residual to a 128-dimensional value space, and a residual decoder that maps the retrieved value back to the latent space. Query and key vectors are ℓ_2-normalized before accessing the memory. The trajectory-specific fast state is therefore a 128×128 matrix and is initialized to zero at the beginning of each trajectory.

The fast-state update rate η is learned jointly with the projection parameters. We parameterize it as

$$\eta = \eta_{\max}\sigma(\alpha).$$

We set $\eta_{\max} = 1.0$ and initialize the resulting update rate to $\eta = 0.5$. Each write operation applies the differentiable delta-rule update described in the main text.

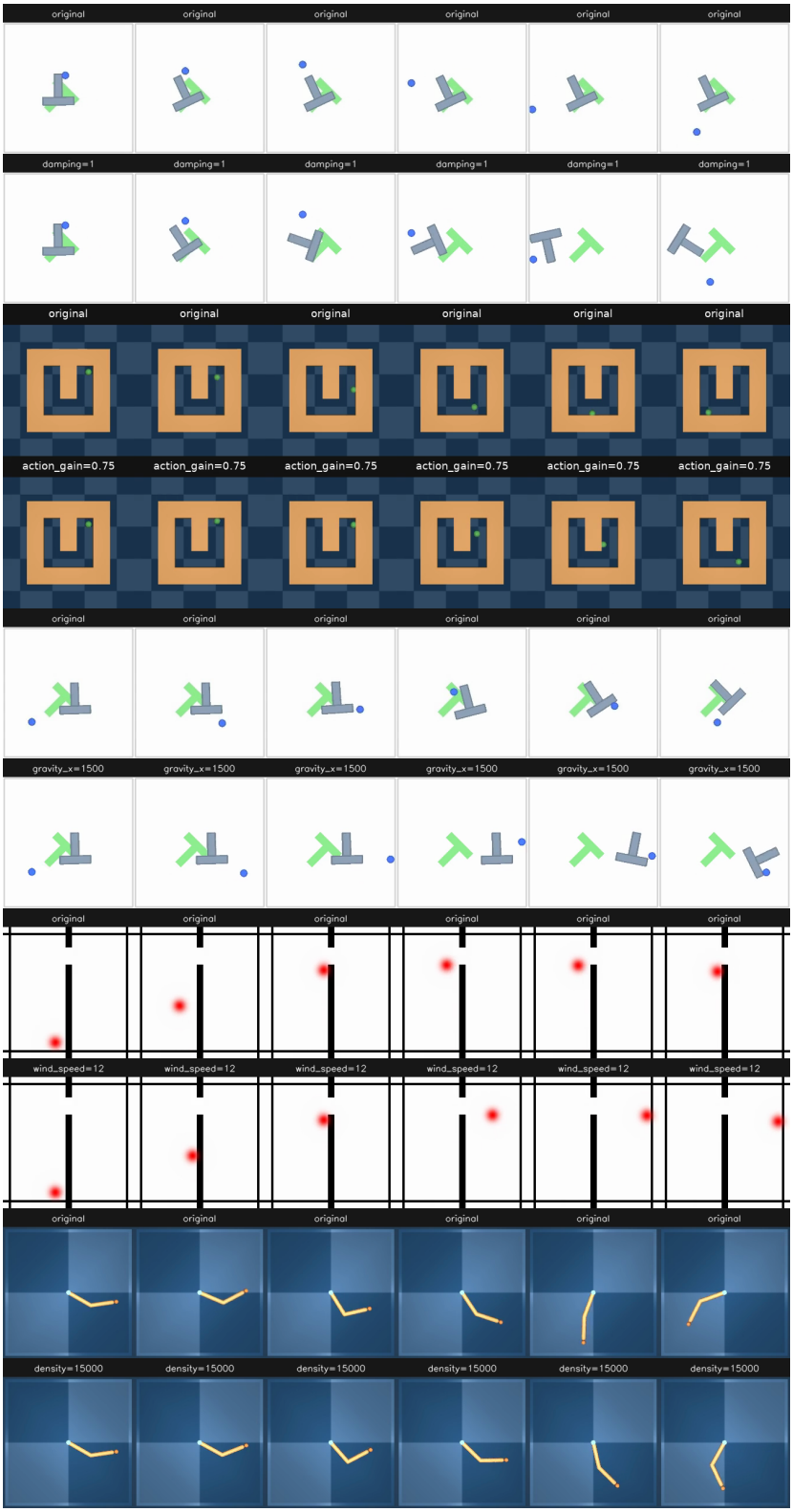


Figure 6: Effect of dynamics interventions under identical actions. For each environment, the same initial state and action sequence are replayed under the original dynamics (top row) and a modified dynamics setting (bottom row). Differences in damping, gravity, action gain, wind speed, and link density cause the resulting observation trajectories to diverge over time despite identical controls.

During meta-training, each sample contains four support transitions followed by one query transition. The fast state is initialized to zero, updated sequentially using residuals from the support transitions, and then held fixed while computing the query prediction loss. Only the query MSE is used to update the shared residual-memory parameters; the pretrained LeWM remains frozen.

At test time, all learned parameters are fixed. For each transition, the model first produces the Base, Adapter-0, and residual-memory predictions using the current fast state; only after the target latent becomes observable is its residual written into memory. This preserves the causal predict–observe–update protocol used throughout evaluation.

C Training Protocol

We train the residual-memory module while keeping the pretrained LeWM encoder, action encoder, and predictor frozen. Each training sample contains a contiguous sequence of

$$H + K + Q$$

steps, where H is the history length required by the pretrained predictor, $K = 4$ is the number of support transitions, and $Q = 1$ is the number of query transitions. The required sequence length is constructed directly from the predictor history size and the configured support/query lengths.

For each batch, observations and actions are first encoded by the frozen pretrained model. The trajectory-local fast state is then initialized to zero independently for every sequence in the batch. The support transitions are processed sequentially. At each support step, we compute the frozen world-model prediction and its realized latent residual, project the corresponding context and residual into a key-value pair, and update the fast state using the delta-rule update.

After all support transitions have been written, the resulting fast state is evaluated on the subsequent query transition without incorporating the query residual into memory. The query prediction is obtained by adding the retrieved residual correction to the frozen world-model prediction. Training minimizes only the query prediction MSE; the inner associative loss produced during support updates is recorded for monitoring but is not added to the optimization objective.

Gradients from the query loss are backpropagated through the differentiable support updates to optimize the shared residual-memory parameters, including the query, key, value, and residual-decoder projections and the learned fast update rate. The pretrained world model receives no gradient updates. We optimize the memory module using AdamW with a learning rate of 10^{-4}, batch size 32, zero weight decay, and gradient-norm clipping at 1.0.

When training and validation use the same dataset collection, we apply a seeded 90/10 split. For parameterized dynamics experiments, up to 100 trajectories are selected per dynamics value, with deterministic sampling controlled by the global random seed. Training and validation loaders use shuffled and non-shuffled sampling, respectively.

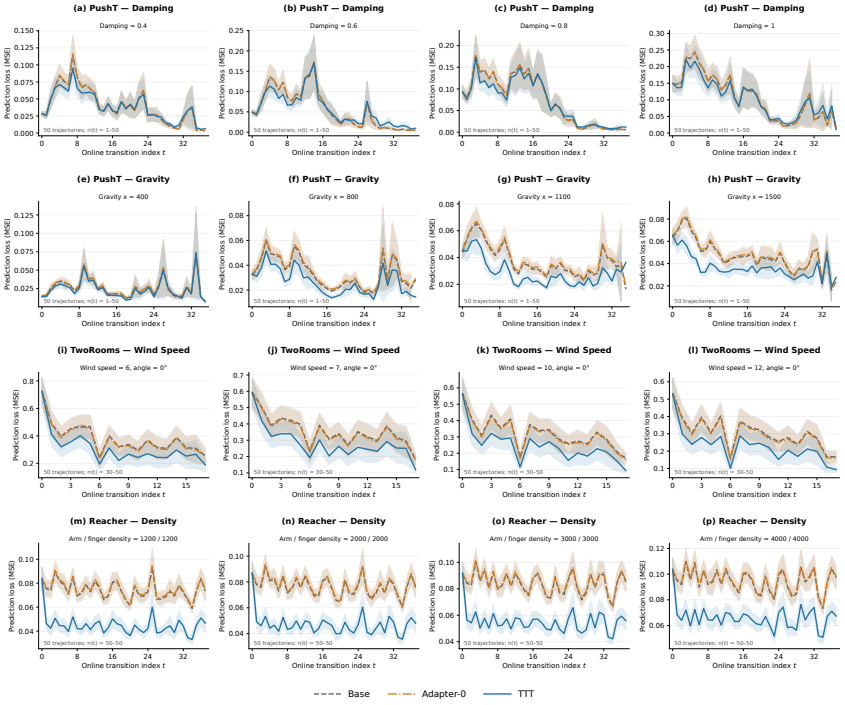
D Additional Online Adaptation Results

D.1 Full Online Adaptation Curves under Held-Out Dynamics Shifts

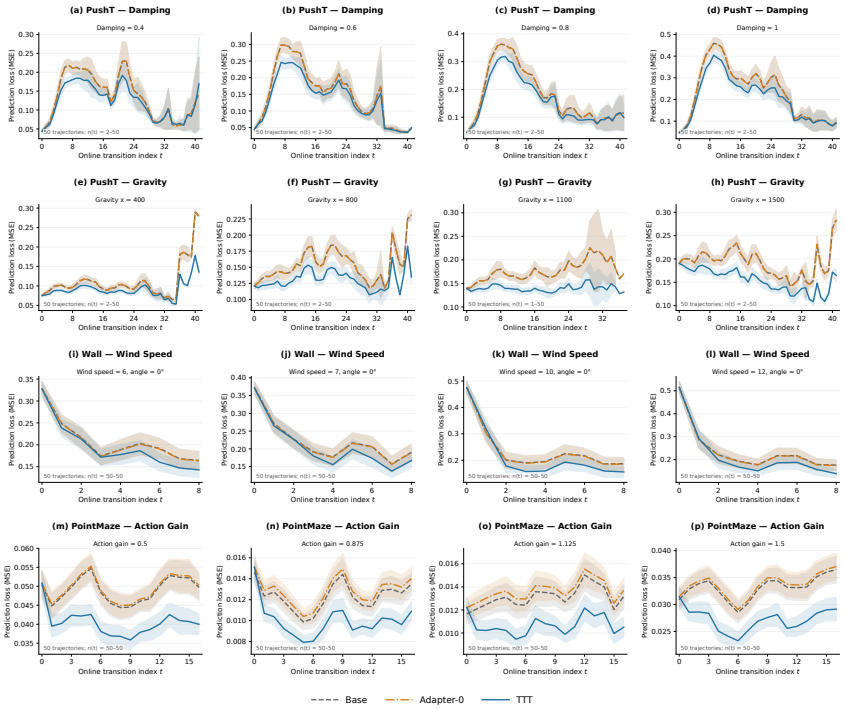
We report the complete per-condition online prediction curves corresponding to the aggregate analysis in the main text. For each held-out dynamics condition listed in Table A.1, we evaluate 50 trajectories and report the one-step latent prediction MSE as a function of the online transition index t. The fast state is initialized to zero at the beginning of each trajectory. Base and Adapter-0 remain fixed throughout evaluation, whereas FRM updates only its trajectory-local fast state after the target latent becomes observable and the corresponding prediction residual can be computed.

Trajectory lengths are not always identical, particularly for PushT. We therefore report $n(t)$, the number of trajectories containing a valid transition at index t, in each panel. Consequently, estimates at later transition indices may be based on fewer trajectories.

Fig. 7 shows the full online adaptation curves for LeWM and DINO-WM across all held-out dynamics settings. Across both backbones and multiple dynamics shifts, FRM generally achieves lower prediction error than Base and Adapter-0, indicating that the aggregate improvements reported in



(a) Lewm online adaptation raw loss



(b) Dino-wm online adaptation raw loss

Figure 7: Full online adaptation curves under held-out dynamics shifts. One-step latent prediction MSE over online interaction for Base, Adapter-0, and TTT across individual test dynamics conditions. Results are shown for (a) LeWM and (b) DINO-WM.

the main text are reflected across individual test conditions rather than being dominated by a small subset of dynamics settings.

D.2 Generalization across Seen and Held-Out Dynamics

We report the complete dynamics sweep for DINO-WM to examine whether the learned residual-memory mechanism generalizes beyond the dynamics values used during meta-training. We use the seen, interpolation, and extrapolation splits defined in Table A.1 across PushT damping and gravity, Wall wind speed, and PointMaze action gain.

At evaluation time, the pretrained world model and learned residual-memory parameters remain fixed, while only the trajectory-local fast state is updated online from observed prediction residuals. Fig. 8 reports the relative TTT gain over the frozen Base predictor for each dynamics value. Positive gains are observed across seen, interpolation, and extrapolation conditions, indicating that the learned write and retrieval mechanism is not restricted to the discrete dynamics values encountered during residual-memory training and can exploit residual feedback under previously unseen dynamics.

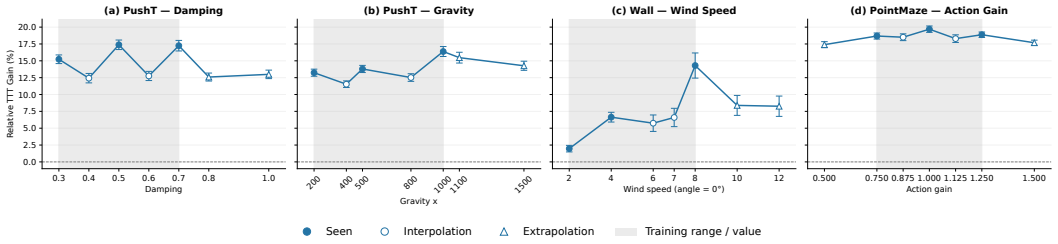


Figure 8: Dynamics generalization of Fast Residual Memory on DINO-WM. Relative TTT gain over Base across seen, interpolation, and extrapolation dynamics conditions.

E Analysis of Residual Reuse

To test whether the gains of Fast Residual Memory can be explained by simply reusing previously observed prediction errors, we evaluate several lightweight residual-reuse strategies on LeWM. All methods keep the pretrained world model frozen and differ only in how past residuals are converted into the correction for the current prediction. We consider the most recent residual (**Last**), the running mean of past residuals (**Mean**), an exponential moving average (**EMA**), and context-based nearest-neighbor retrieval (**kNN**).

Table E.1: Comparison of adaptation strategies under held-out dynamics shifts. Lower prediction loss is better; higher relative gain is better. Best results are shown in bold.

Method	**PushT–Damping**		**PushT–Gravity**		**TwoRooms–Wind**		**Reacher–Density**	
	Loss ↓	Rel. gain ↑	Loss ↓	Rel. gain ↑	Loss ↓	Rel. gain ↑	Loss ↓	Rel. gain ↑
Base	0.1011	–	0.0414	–	0.3495	–	0.0822	–
Last	0.1344	-19.11%	0.0443	-6.38%	0.5012	-26.21%	0.0745	12.42%
Mean	0.1142	-12.43%	0.0409	3.80%	0.3564	4.12%	0.0648	22.05%
EMA	0.1025	2.15%	0.0363	13.91%	0.3045	18.26%	0.0600	27.96%
kNN	0.1159	-7.36%	0.0392	7.68%	0.3478	8.73%	0.0575	31.70%
TTT	**0.0930**	**8.71%**	**0.0330**	**20.63%**	**0.2720**	**31.15%**	**0.0560**	**32.65%**

The results show that residual feedback does contain reusable predictive structure, but that naive reuse is not uniformly reliable. Directly applying the most recent residual degrades performance in three of the four environments, indicating that adjacent prediction errors are not generally interchangeable. Aggregating residuals is generally more effective. EMA yields positive relative gains

across all four settings under our reported relative-gain metric, with particularly large improvements in PushT Gravity, TwoRooms Wind, and Reacher Density. On PushT Damping, however, the aggregate MSE remains slightly above Base (0.1025 vs. 0.1011), indicating that the benefit of temporal smoothing is much weaker in this environment. Overall, these results support the basic premise that prediction residuals contain reusable test-time information, while also showing that their usefulness depends on the underlying dynamics.

At the same time, the effectiveness of a fixed residual-reuse rule varies substantially across dynamics. Mean and kNN provide large improvements in Reacher but are much less effective, or even harmful, in PushT Damping. In particular, kNN shows that retrieving residuals from similar past contexts can be highly effective in some environments, but context similarity alone does not guarantee that the retrieved residual is predictive for the current transition. In contrast, Fast Residual Memory achieves positive gains across all four LeWM dynamics families in the main evaluation. These results suggest that its benefit does not arise from residual reuse alone; rather, learning how residual information should be associated with and retrieved for the current predictive context provides a more consistent mechanism across heterogeneous dynamics shifts.