ADAJEPA: AN ADAPTIVE LATENT WORLD MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent world models enable planning from high-dimensional observations by predicting future states in a compact latent space. However, these models are typically kept frozen at test time: when their predictions become inaccurate, planning can fail, especially under test-time distribution shift. To address this, we propose AdaJEPA, an adaptive latent world model that performs test-time adaptation within the closed loop of model predictive control (MPC). After training, AdaJEPA plans and executes the first action chunk, uses the observed next-state transition as a self-supervised adaptation signal, and replans with the updated model. This closed-loop update continuously recalibrates the world model without additional expert demonstrations. Across a range of goal-reaching tasks, AdaJEPA substantially improves planning success with as few as one gradient step per MPC replanning step.

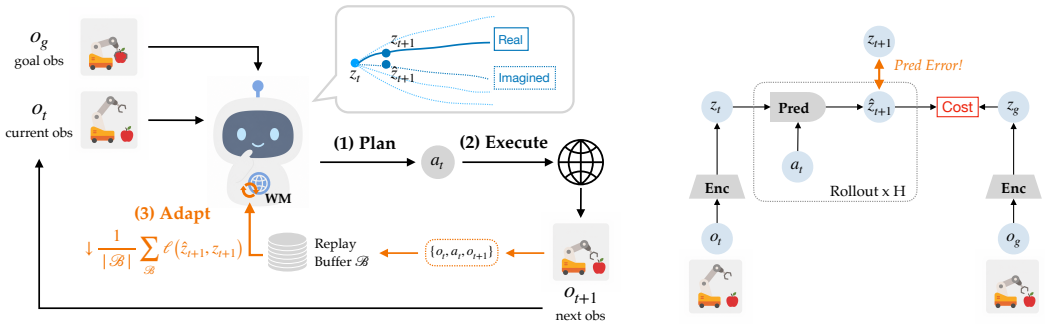
1 INTRODUCTION

The long-standing goal of latent world models is to capture environment dynamics in a compact latent space that enables efficient prediction and planning with better generalization. Joint-Embedding Predictive Architectures (JEPAs) have emerged as a powerful world model paradigm that jointly learns an encoder and a predictor by optimizing a latent prediction objective on reward-free offline trajectories (LeCun, 2022; Sobal et al., 2025; Assran et al., 2025; Zhou et al., 2025). Within this framework, the planning task is defined at test time and is often performed with model predictive control (MPC) (García et al., 1989; Kouvaritakis & Cannon, 2016), which repeatedly rolls out the model forward, optimizes a short-horizon action sequence, executes the first (or first few) action(s), and replans from the next observation. This combination of world models with MPC has become a standard recipe for goal-conditioned control and planning.

Yet, the world model used for planning is typically frozen after training. When its predictions are inaccurate, MPC optimizes the wrong objective: actions that appear effective in latent rollouts may fail under the true system dynamics. Even small one-step errors can compound over the planning horizon, causing actions that appear effective in latent imagination to fail in the real environment. This issue is amplified under test-time distribution shift: visual changes such as noise, lighting, or background distractors can misalign the encoder, while changes in friction, mass, or contact dynamics can misalign the latent predictor. As a result, even high-capacity world models can suffer substantial performance degradation when small changes occur in test environments (Zhou et al., 2025; Toso et al., 2026), which hinders the application of world models in the real world. This suggests that *a world model should not remain fixed after training, but continue to improve from the experience encountered during deployment*. This intuition is well-grounded in biological systems, where sensorimotor adaptation relies on cerebellar mechanisms to adjust behavior under changing inputs and dynamics (Shadmehr & Mussa-Ivaldi, 1994; Wolpert et al., 1998; Bastian, 2006; Shadmehr et al., 2010). We humans also continually update internal world models with new experience, and these updates shape subsequent decisions (Craik, 1943; Gläscher et al., 2010; Nassar et al., 2010; Daw et al., 2011).

To adapt world models during deployment, we propose **AdaJEPA**, a test-time adaptation framework that operates within the closed loop of MPC. Rather than treating the model as fixed after training, AdaJEPA uses the transition observed after each executed action as a self-supervised signal to adapt the world model before the next replan (Figure 1). This makes learning and planning tightly coupled: the model optimizes actions using its predictions, the consequence of the actions provides the adaptation signal, and the adapted model improves prediction and thus planning for the next iteration. The adaptation is both sample- and compute-efficient, as each update modifies only a small

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.



(a) AdaJEPA: Plan-Act-Adapt-Replan Loop

(b) World Model for Planning

Figure 1: AdaJEPA performs test-time adaptation during closed-loop MPC. At each MPC step, we plan with the current model, execute the first action a_t, collect observation o_{t+1} from the environment, and update the model to minimize the prediction error on the newly observed transition $\{o_t, a_t, o_{t+1}\}$ before replanning. This yields a simple planexecuteadaptreplan loop that continually recalibrates the model to transitions encountered in the current environment.

subset of parameters and can be performed with a single gradient step on the latest transition of the current episode. AdaJEPA significantly outperforms the frozen model across in-distribution and out-of-distribution goal-reaching tasks, with particularly strong gains when training data is limited.

2 Related Work

JEPA world models. Joint-Embedding Predictive Architectures (JEPAs) encode high-dimensional sensory observations into a compact latent space and learn dynamics within it. They consist of an encoder and a predictor which are typically trained jointly by minimizing a latent prediction objective over reward-free offline trajectories (LeCun, 2022; Zhou et al., 2025; Sobal et al., 2025; Assran et al., 2025; Goswami et al., 2025; Wang et al., 2026; Maes et al., 2026; Zhang et al., 2026). Once trained, these models are paired with planners such as MPC to solve goal-conditioned tasks. However, all of the aforementioned world models are kept frozen at test time, leading to potential performance degradation under test-time distribution shift. Closest in motivation, Parthasarathy et al. (2025) aim to reduce the train-test gap through data synthesis during training, but their model still remains fixed during planning and is evaluated only on in-distribution environments. To the best of our knowledge, we are the first to adapt a JEPA world model during planning.

Test-time training and adaptation. Test-time training/adaptation (TTT/TTA) updates a pretrained model using test data to reduce test-time distribution shift. Unlike supervised finetuning on a labeled target dataset, TTT/TTA typically optimizes an unsupervised or self-supervised objective computed from the test input itself. Early applications mainly focus on image classification with common objectives including auxiliary tasks (Sun et al., 2020), entropy minimization (Wang et al., 2021; Niu et al., 2022; Wang et al., 2022; Niu et al., 2023), augmentation-based consistency (Zhang et al., 2022), and masked reconstruction (Gandelsman et al., 2022). The same principle has since been explored in a wide range of domains, including in-context learning (Akyürek et al., 2025) and embodied decision making (Hong et al., 2026) with LLMs, prompt tuning of vision-language models (Yoon et al., 2024), and streaming videos (Wang et al., 2025b). Our work applies test-time adaptation to latent planning with world models: each execution leads to the next observation, which serves as a self-supervised prediction target for adaptation.

Adaptation in planning and control. The idea of updating predictive models during decision making dates back to adaptive control: for example, self-adapting IDCOM identifies a low-dimensional input–output process model online and uses it to retune the predictive controller (Foigel & Richalet, 1979). Similarly, online model-based RL updates dynamics models from interaction, but these updates are usually coupled to policy or value learning (Sutton, 1991; Thrun et al., 1990; Hansen et al., 2022; 2024). Recently, several world models also focus on adapting pretrained predictors, but typically require target-domain finetuning data, additional online rollouts, or outer self-improvement loops (Wang et al., 2025a; Gao et al., 2025; Lanier et al., 2025; Luo et al., 2026).

AdaJEPA instead adapts a pretrained world model inside closed-loop MPC: each executed transition provides a self-supervised latent prediction target, and the updated world model is reused at the next replan, *without reward labels, expert labels, or a separate data-collection phase*.

3 ADAJEPA: AN ADAPTIVE LATENT WORLD MODEL

Unlike existing world models that are kept frozen during planning, AdaJEPA performs test-time adaptation during closed-loop MPC. At test time, we plan with the current model, execute the first action, and update the model using the newly observed transition before replanning. This yields a simple plan–execute–adapt–replan loop that continually recalibrates the model to transitions encountered during planning.

3.1 BACKGROUND: JEPA WORLD MODELS

We consider trajectories of high-dimensional observations $o_t \in \mathbb{R}^{n_o}$ and actions $a_t \in \mathbb{R}^{n_a}$. A latent world model consists of a sensory encoder $\mathcal{E}^s_\phi$, an action encoder $\mathcal{E}^a_\psi$, and a predictor f_θ:

$$z_t = \mathcal{E}^s_\phi(o_t), \qquad u_t = \mathcal{E}^a_\psi(a_t), \qquad \hat{z}_{t+1} = f_\theta(z_t, u_t). \tag{1}$$

f_θ may be conditioned on a short history of latent states and action embeddings, and we use the one-step notation for simplicity. The encoder and predictor are jointly trained on a reward-free offline transition data $\mathcal{D}_{\text{off}} = \{(o_t, a_t, o_{t+1})\}$ by predicting future latent targets rather than reconstructing pixels. A generic JEPA-style prediction objective is

$$\mathcal{L}_{\text{pred}} = \frac{1}{K} \sum_{k=1}^{K} \ell(\hat{z}_{t+k}, z_{t+k}), \tag{2}$$

where z_{t+k} is the target representation of o_{t+k} and ℓ is a latent prediction loss such as MSE. Different JEPA instantiations may either use the stop-gradient operator (Chen & He, 2021) on the target representation or include a regularization term (Bardes et al., 2022; Balestriero & LeCun, 2025) in the training objective to prevent collapse.

After training, the world model can be used for goal-conditioned latent planning. Given a goal observation o_g with latent representation $z_g = \mathcal{E}^s_\phi(o_g)$, MPC optimizes an action sequence by rolling out f_θ and minimizing a latent goal-reaching cost (Figure 1b):

$$a^*_{t:t+H-1} = \arg\min_{a_{t:t+H-1}} \sum_{k=1}^{H} \alpha_k \, d(\hat{z}_{t+k}, z_g), \tag{3}$$

where H is the planning horizon, α_k are temporal weights, and d is typically the squared Euclidean distance, $d(\hat{z}, z_g) = \|\hat{z} - z_g\|_2^2$. The optimization in equation 3 can be solved with either gradient-based planners or sampling-based methods such as CEM (Rubinstein, 1997). Standard MPC executes the first action a_t, after which the agent observes the next observation o_{t+1} and replans.

3.2 CLOSED-LOOP PLAN-AND-ADAPT

A pretrained world model is never perfect. Prediction errors can arise from finite offline data and test-time distribution shifts. To minimize prediction errors at deployment and thus improve planning success, AdaJEPA continuously updates itself using the transitions caused by its own actions.

Algorithm 1 details the proposed *plan-execute-adapt-replan* loop (visualized in Figure 1a) for one episode. At each MPC replanning step, we plan with the current model, execute the first action, observe the resulting transition, perform a small number of self-supervised updates, and replan with the updated model. Note that each episode starts from the same pretrained model and maintains its own copy and buffer throughout the episode. We describe the key components of the test-time adaptation in the following.

Online buffer. The buffer $\mathcal{B}$ stores recent transitions collected during MPC. After executing a_t and observing o_{t+1}, we append (o_t, a_t, o_{t+1}) to $\mathcal{B}$. Because $\mathcal{B}$ can grow unbounded, we cap it to a fixed size N at each iteration. We consider two strategies: (i) `recent-N` keeps the most recent N transitions, and (ii) `hard-N` keeps only the N transitions that result in the largest prediction errors (comparison is in Appendix B.4).

Algorithm 1 AdaJEPA: Closed-Loop Plan-and-Adapt

Input: pretrained world model $(\mathcal{E}^s_\phi, \mathcal{E}^a_\psi, f_\theta)$, trainable parameters Ω, goal observation o_g, planning horizon H, adaptation steps U, buffer size N, max steps T
Initialize buffer $\mathcal{B} \leftarrow \emptyset$; observe o_0
for $t = 0, 1, \ldots, T-1$ **do**
    Plan with the current model: optimize an action sequence by minimizing a latent goal-reaching cost (eq equation 3)
    Execute the first action a_t and observe o_{t+1}
    Add (o_t, a_t, o_{t+1}) to $\mathcal{B}$ and trim (if necessary) to keep N transitions at maximum
    for $u = 1, \ldots, U$ **do**
        $\Omega \leftarrow \Omega - \eta \nabla_\Omega \mathcal{L}_{\text{ada}}(\mathcal{B})$
    end for
end for

Adaptation loss. AdaJEPA uses the same self-supervised prediction signal at test time as in pretraining. For clarity, we assume one-step history and no frameskip. With the replay buffer $\mathcal{B}$, the adaptation loss is

$$\mathcal{L}_{\text{ada}}(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{(o_i, a_i, o_{i+1}) \in \mathcal{B}} \ell\big(f_\theta\big(z_i, \mathcal{E}^a_\psi(a_i)\big), \text{sg}(z_{i+1})\big), \tag{4}$$

where $z_i = \mathcal{E}^s_\phi(o_i)$, ℓ is a latent prediction loss, and $\text{sg}(\cdot)$ denotes the stop-gradient operator. Here, we use stop-gradient as the default anti-collapse stabilizer during online adaptation, though it can be replaced by other methods depending on the pretrained world model. Removing stop-gradient while updating only the last layers of the encoder and predictor for one step gives similar planning performance, suggesting that the restricted online update already limits collapse. For longer histories, action chunks, and frameskips, we average the same loss over all valid prediction windows.

Adapted parameters. Let $\Omega \subseteq \{\phi, \psi, \theta\}$ denote the parameters updated at test time. After each MPC step, AdaJEPA performs U gradient updates,

$$\Omega \leftarrow \Omega - \eta \nabla_\Omega \mathcal{L}_{\text{ada}}(\mathcal{B}). \tag{5}$$

The adapted model is immediately reused for the next planning problem. In experiments, Ω can be restricted to a small subset of encoder or predictor parameters, making adaptation lightweight.

4 Experiments

In this section, we evaluate the planning performance of our proposed AdaJEPA on different train–test distribution shifts and under different adaptation modes and design choices. Our results demonstrate that adaptation with only one GD step in each MPC replanning step can lead to strong and robust performance gains in all settings.

4.1 Setup

Environments. Our main results are conducted on the PushT (Chi et al., 2025) and PointMaze (Fu et al., 2021) benchmarks (environment details in Appendix A). We evaluate planning on unseen in-distribution trajectories, as well as on the following out-of-distribution variations:

- *Shape shifts* change the block in the PushT environment from a T to other shapes following the PushObj setup in (Zhou et al., 2025), but keep the colors unchanged. We train on four shapes {`T`, `L`, `Z`, `+`} and test on both the training shapes and three held-out OOD shapes {`I`, `smallT`, `square`} that share the same contact dynamics but have unseen geometry. To ensure a reasonable contact ratio, we bias the training and testing data toward contact-rich interactions, forcing at least one contact in the test trajectory. Goals are sampled 25 steps away. Examples of shapes are in Figure 3 and more details on data generation are in Appendix A.2.
- *Visual shifts* apply per-frame corruptions to the original PushT observations by adding (i) Gaussian blur, (ii) salt-and-pepper (snp) noise, and (iii) dark lighting. Additionally, we test the robustness to color shifts by changing (iv) the moving T from light gray to red, (v) the

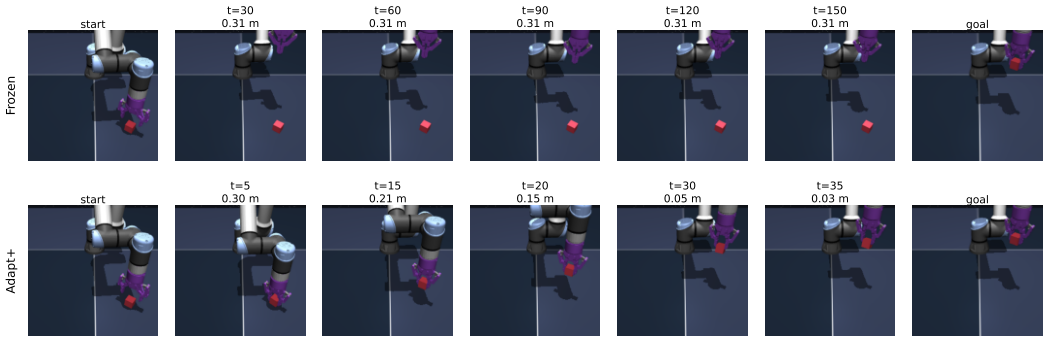


Figure 2: OGBench-Cube Example: a task the frozen model fails (upper) and AdaJEPA succeeds (lower). Frames at the env steps shown, with the cube-to-goal distance underneath (success requires under 0.04 m); the goal frame is on the right.

anchor T from light green to red, and (vi) the agent from blue to red. The model is trained on the original PushT data but tested on these OOD settings. Goals are sampled 25 steps away. Examples are in Figure 3 and more details of data generation are in Appendix A.2.

- *Dynamics shifts* vary physics in the PointMaze-Medium environment. We change physics to (i) low mass ($\times 0.1$) which causes the agent to move faster under the same force, and (ii) high damping ($\times 50$) that causes velocity to decay faster. The model is trained on the default dynamics using the training data from Wang et al. (2026) and tested on these OOD dynamics settings. Goals are sampled such that their grid distance from the start is larger than 3. More details are in Appendix A.3.
- *Layout shifts* vary the maze layouts in the PointMaze environment as proposed by (Sobal et al., 2025; Zhang et al., 2026). Maze layouts are randomly generated on an 8×8 grid with connected free space. We train on 25 maze layouts and evaluate on 5 held-out layouts. At test time, goals are sampled such that the shortest-path distance from the start is 3–5 cells (easy) or 6–9 cells (hard), ensuring nontrivial but feasible navigation tasks. More details are in Appendix A.4.

Plan-and-Adapt. We use receding-horizon MPC for all experiments, with either GD or CEM as the trajectory optimizer. Unless specified otherwise, AdaJEPA performs adaptation at every MPC replanning step by updating only the final layers of the visual encoder and predictor, while keeping all other parameters frozen. Updates use the same learning rates as training ($\eta_{\text{pred}} = 5 \times 10^{-4}$ and $\eta_{\text{enc}} = 10^{-5}$) under one of two schedules: (i) **Adapt**: one gradient step per replan, (ii) **Adapt+**: up to five steps per replan, taken only while the buffer loss exceeds $0.1\times$ the copy-baseline loss (predicting no change). We maintain a replay buffer containing the 5 most recent samples. At each MPC replanning step, only a single action chunk is executed, and the maximum number of MPC steps is set to 30. All results are averaged over three test-data seeds, with 50 episodes per seed.

Architectures. For each environment, we train a JEPA world model from scratch, following (Wang et al., 2026). The model consists of a ResNet encoder that produces global features and a transformer-based predictor. We apply stop-gradient to the target branch in the prediction loss to prevent collapse and use curvature regularization to encourage straighter latent trajectories to facilitate planning. Following prior work (Zhou et al., 2025; Wang et al., 2026), action embeddings are concatenated with visual and proprioceptive embeddings before being passed to the predictor. We use a frameskip of 5 and a history window of 3. Implementation details are provided in Appendix B. Note that AdaJEPA is *agnostic to the underlying world model implementation*, and we show consistent improvements across model variants in Table 2.

4.2 Results

In-distribution performance. We first examine how AdaJEPA behaves on test environments same as training. On PushObj training shapes and the default PushT rendering, test-time adaptation

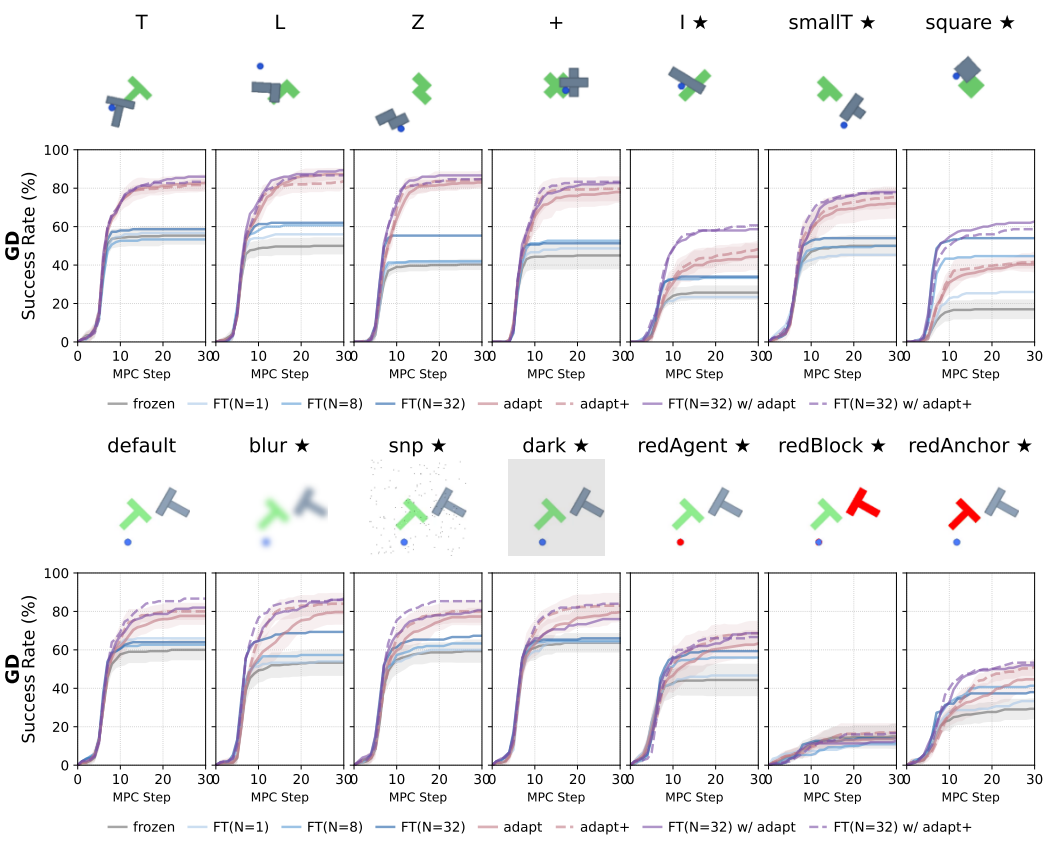


Figure 3: Planning Success under Shape Shifts (top) and Visual Shifts (bottom). The ★ denotes unseen shapes and configurations. Here, we use GD sub-planner and the CEM sub-planner gives a similar trend (Figure 7). FT(N) is the frozen model fine-tuned offline on N target-domain trajectories (the parameters AdaJEPA updates, one epoch, frozen during planning). AdaJEPA consistently improves planning success across all settings, with comparison between frozen and AdaJEPA planning trajectories in Appendix C.

significantly improves over the frozen model for both GD (Figure 3) and CEM planners (Figure 7). As PushObj model is trained on various shapes, adaptation to the current shape makes the model specialize and results in a performance boost from 40–55% to 78–87%. The gain is also consistent in the model trained with PushT only, across three implementations in Table 2. We also test an in-distribution robotic-arm manipulation task, the OGBench cube, where Adapt and Adapt+ consistently improve the performance (Appendix B.6); Figure 2 shows a task the frozen model fails and AdaJEPA solves. In summary, test-time adaptation is safe to apply in-distribution: it yields large gains when the frozen model is suboptimal and stays within noise of it when it is already near-optimal. Adapt+ further improves with its gate taking no update when the model already predicts well and spends its updates only where the prediction error is large.

Distribution shifts. Figure 3 and table 1 report planning success under the four families of shifts, where AdaJEPA improves over the frozen model in nearly all settings. Under shape and visual shifts (Figure 3), the success of the frozen model saturates within the first few replanning steps, whereas AdaJEPA continues to improve throughout the planning horizon, indicating that adaptation enables the planner to complete tasks on which the frozen model stalls. On unseen shapes, the success rate nearly doubles. Under dynamics and layout shifts (Table 1), the frozen model already performs well under altered dynamics, likely because the three-frame history provides a form of in-context adaptation, yet AdaJEPA yields further gains, most notably under high damping. On unseen layouts, AdaJEPA improves success from 65% to 84% with GD, with comparable gains on harder goals, and the resulting trajectories follow the shortest path more closely (Figure 4). Since our adaptation is lightweight, updating only the final layers from a handful of transitions, it cannot compensate for

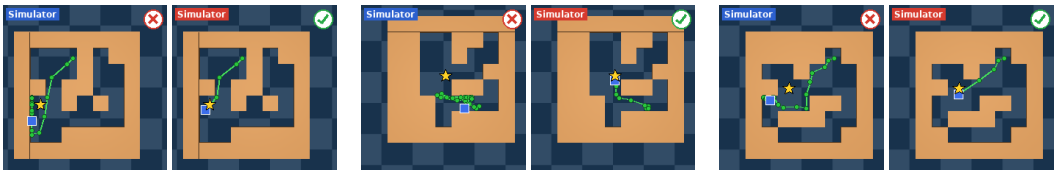
Table 1: Planning Success under Dynamics and Layout Shifts for PointMaze. Note that low mass causes faster movement under same force and high damping causes velocity decay faster. Adapt and Adapt+ updates the frozen model online during the episode; the fine-tuning rows finetune the frozen model offline on N target-domain episodes before planning and are frozen during planning

.

	Dynamics Shift (PointMaze-Medium)				Layout Shift			
	mass $\times 0.1$		damping $\times 50$		Unseen Layouts (easy)		Unseen Layouts (hard)	
Method	GD	CEM	GD	CEM	GD	CEM	GD	CEM
Frozen	77.3 ± 5.8	83.3 ± 5.2	52.7 ± 3.6	60.0 ± 4.8	64.7 ± 0.5	54.5 ± 1.3	31.5 ± 2.7	26.1 ± 2.2
Adapt	77.0 ± 4.7	85.3 ± 2.2 $\uparrow 2.0$	61.0 ± 7.5 $\uparrow 8.3$	67.0 ± 1.9 $\uparrow 7.0$	78.8 ± 3.4 $\uparrow 14.1$	70.0 ± 0.0 $\uparrow 15.5$	39.1 ± 2.9 $\uparrow 7.6$	34.5 ± 2.8 $\uparrow 8.4$
Adapt+	79.7 ± 5.0 $\uparrow 2.3$	86.7 ± 1.9 $\uparrow 3.3$	75.0 ± 5.6 $\uparrow 22.3$	76.3 ± 4.2 $\uparrow 16.3$	83.9 ± 2.0 $\uparrow 19.2$	77.6 ± 2.8 $\uparrow 23.1$	50.0 ± 2.7 $\uparrow 18.5$	42.4 ± 2.6 $\uparrow 16.3$
Fine-tuning on target-domain data								
$N = 1$	80.7 ± 4.7	82.0 ± 4.3	55.3 ± 1.9	58.7 ± 2.5	63.1 ± 2.2	52.7 ± 0.7	30.4 ± 1.2	25.1 ± 0.8
$N = 8$	78.0 ± 0.0	85.3 ± 3.4	60.7 ± 3.4	58.0 ± 3.3	57.2 ± 3.3	48.1 ± 2.1	25.3 ± 0.4	23.9 ± 1.5
$N = 32$	73.3 ± 5.2	78.7 ± 0.9	69.3 ± 4.1	63.3 ± 3.8	60.4 ± 0.9	50.3 ± 2.0	33.1 ± 0.5	25.3 ± 3.7



(a1) Default frozen → success (a2) Low mass frozen → failure (a3) Low mass adapt → success (b1) Default frozen → success (b2) High damp frozen → failure (b3) High damp adapt → success



(c1) Maze 1 frozen → failure (c2) Maze 1 adapt → success (d1) Maze 2 frozen → failure (d2) Maze 2 adapt → success (e1) Maze 3 frozen → failure (e2) Maze 3 adapt → success

Figure 4: Planning trajectories under dynamics shifts (top, PointMaze-Medium) and layout shifts (bottom, unseen mazes). Green: the agent's path; blue square: final position; gold star: goal. The frozen model succeeds under the training dynamics but fails under low mass, high damping, and on unseen layouts, where AdaJEPA recovers success with shorter paths.

shifts that the representation itself fails to capture. Under the red-block shift, the frozen encoder no longer encodes the block's orientation, and no update of the adapted parameters recovers it. Fine-tuning the full encoder on target-domain data does, and AdaJEPA provides additional gains on top (Appendix B.3).

TTA vs. FT. How does AdaJEPA compare to simply fine-tuning (FT) on target-domain data? We fine-tune the same parameters that AdaJEPA updates, with the same learning rates, on $N \in \{1, 8, 32\}$ target-domain trajectories for one epoch, and plan with the fine-tuned model frozen. A trajectory has 100 (PointMaze) to 230 (PushT) steps, comparable to the 150 steps one planning episode generates (Figure 3 and table 1). AdaJEPA is more *sample efficient* than FT Given a single trajectory, FT leaves the frozen model almost unchanged on every shift, whereas AdaJEPA improves substantially across setups. AdaJEPA outperforms the finetuning baselines even with much more data (32 trajectories) on most setups. The two are also complementary. AdaJEPA applied on top of the fine-tuned model further boosts the performance on the hardest shifts such as the unseen square in Figure 3 and redBlock in Appendix B.3.

Latency. While the adaptation step significantly increases the planning success rate, it inevitably increases latency due to extra parameter update. The added time is 0.01–0.03 s per replanning step (Table 2), because only a small subset of parameters takes a single step. Moreover, adaptation also makes the agent reach the goal in equal-or-fewer replans, thus decreasing the overall time. These results show that AdaJEPA's lightweight test-time adaptation is both efficient and effective.

Table 2: AdaJEPA across different implementations. The planning success is evaluated using PushT validation trajectories from (Zhou et al., 2025). The reported values are the average MPC success rate (%) and per-replan time using one H200. Test-time adaptation consistently improves and introduces almost negligible latency.

Encoder / Predictor	Latent Dim	Anti-collapse	Setting	GD (%)	Time (s)	CEM (%)	Time (s)
WM w/ Temporal Straightening (global feat.) (Wang et al., 2026)	1×384	stop-grad	Frozen	84.0 ± 2.0	3.14	74.0 ± 3.5	0.24
			Adapt	85.3 ± 3.1 ↑1.3	3.17 ↑0.03	81.3 ± 6.4 ↑7.3	0.27 ↑0.03
WM w/ Temporal Straightening (spatial feat.) (Wang et al., 2026)	14×14×8	stop-grad	Frozen	91.3 ± 4.2	3.37	89.3 ± 3.1	5.37
			Adapt	92.0 ± 3.5 ↑0.7	3.38 ↑0.01	93.3 ± 2.3 ↑4.0	5.39 ↑0.02
DINO-WM (patch) (spatial feat.) (Zhou et al., 2025)	14×14×384	-	Frozen	68.0 ± 10.6	3.66	86.7 ± 6.1	9.53
			Adapt	70.0 ± 4.0 ↑2.0	3.68 ↑0.02	90.0 ± 3.5 ↑3.3	9.56 ↑0.03

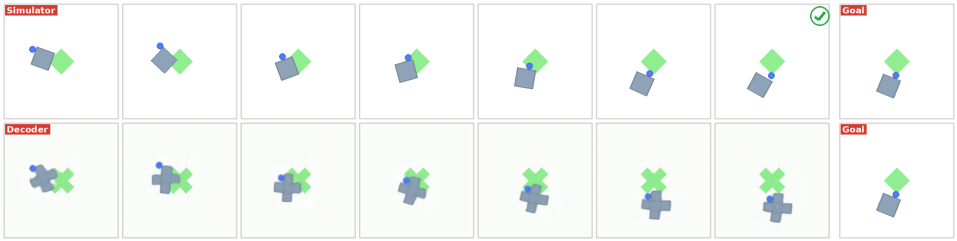


Figure 5: Example AdaJEPA Planning under a Shape Shift. The decoder is trained on pretrained representations, i.e. PushObj data which only includes {T, L, Z, +} while square is unseen.

Visualization. To better understand what test-time adaptation changes in the world model, we visualize imagined rollouts after adaptation. We train a decoder on the pretrained latent space and find it can still reconstruct visual rollouts after light-weight test-time adaptation. Under visual corruptions or unseen PushObj shapes, decoded rollouts often retain training-domain structure: for example, an unseen object may be decoded as a visually similar seen shape (Figure 5), and an unseen red PushT block as the training color (Appendix C). This suggests that AdaJEPA improves planning by exploiting shared latent structure and recalibrating predictions, while remaining close to the learned latent manifold.

4.3 Ablations

Which parameters to adapt? We compare direct updates to selected predictor and encoder layers and LoRA updates to the full model (full results are in Figure 8). Across shifts, all adaptation choices improve over the frozen model, indicating that the superior performance of AdaJEPA is not tied to a specific adaptation target. For shape shifts, all variants perform similarly, even when the encoder is frozen, suggesting that the pretrained representation shows generalization across object geometries and that most of the needed correction lies in the predictor. For visual and layout shifts, updating only the predictor is less effective, as the mismatch enters through the observation representation and cannot be fully corrected with predictor alone. Interestingly, adapting the first layer of the predictor is particularly effective to Layout Shifts, likely because this layer is closest to the latent and action inputs and can better recalibrate local transition structure under new maze connectivity. LoRA also improves over the frozen model, but does not consistently outperform direct updates to selected layers. Overall, the best adaptation target is *environment-dependent, but performance is not highly sensitive*: simple selected-layer updates, such as adapting the first or last predictor block together with the last encoder stage, are already strong across settings.

How should adaptation hyperparameters be chosen? Since AdaJEPA adapts inside the MPC loop, the update must be effective with little tuning and minimal latency. We ablate the test-time learning rate, number of gradient steps, and replay buffer in Figure 9. *A single gradient step with the training learning rate provides a strong practical default*: larger learning rates can make one-step updates effective but become less stable with more steps, while smaller learning rates often require additional updates and increase replanning cost. Replay buffer design has a smaller effect: AdaJEPA improves over the frozen model across buffer choices, including no buffer, with a recent-transition

buffer giving the most stable gains. For out-of-distribution settings, stronger adaptation—via a larger learning rate, more optimization steps, or a larger recent replay buffer—is often beneficial when latency permits. These hyperparameters can be tuned for a target environment if necessary, while the default provides a practical starting point.

AdaJEPA improves over various JEPA world models. We evaluate AdaJEPA across multiple JEPA world-model implementations on the PushT validation trajectories (details in Appendix A.1). As shown in Table 2, *AdaJEPA improves planning success across global and spatial representations, model architectures, GD and CEM planners, and different training objectives*. Even though these models are well trained and evaluated in distribution, test-time adaptation still yields consistent gains while adding only 0.01–0.03s per MPC replanning step. These results indicate that AdaJEPA is broadly efficient and effective for improving the test-time performance for latent world models.

4.4 Discussion

The experiments above show that AdaJEPA consistently improves planning by correcting prediction errors during closed-loop MPC, and is robust across environments, planners, base models, and adaptation targets. To further understand the strengths and limitations of AdaJEPA, we study how training data scale affects generalization and adaptivity in this section.

We analyze two axes: shape diversity K and trajectories per shape N, and report the average success on seen and unseen shapes of PushObj across different scales in Figure 6, with more experiment details in Appendix B.5. In general, larger and more diverse training data improves both frozen and AdaJEPA across seen and unseen shapes. Shape diversity is especially important for generalization before and after adaptation: for example, under the same 16k total trajectory budget, distributing data across four shapes (K=4, N=4k) achieves 51.9% unseen success with AdaJEPA, compared with 45.8% when all trajectories come from a single shape (K=1, N=16k). Test-time adaptation improves success rates across scales, especially in *low-data* regimes: on seen shapes with K=1, N=1k, it improves success from 28.1% to 60.8%, more than doubling performance over the frozen model and surpassing a frozen model trained with 16× more trajectories per shape (43.5%). Thus, while more training data improves generalization, *test-time adaptation can compensate for limited training coverage* by refining the model during deployment, providing a more sample-efficient route that is complementary to data scaling.

As we only perform lightweight correction during planning, its effectiveness is bounded by the coverage of the pretrained representation: when the test environment requires features absent from training, adaptation can improve planning but may not fully close the gap. A natural next step is to combine lightweight test-time adaptation with continual and active learning to expand the world model's coverage over time.

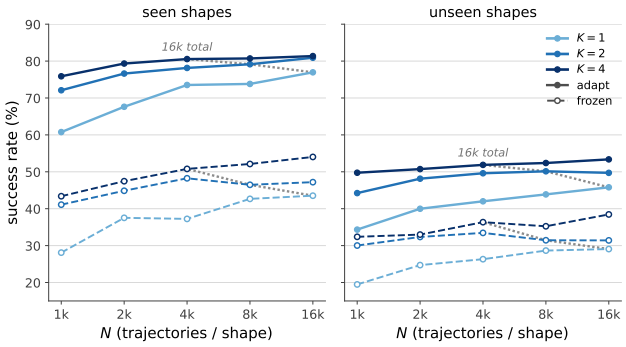


Figure 6: Effect of training data scale on PushObj planning success: shape diversity K and trajectories per shape N.

5 Conclusion

We introduced AdaJEPA, an adaptive world model that performs test-time adaptation during the closed-loop MPC. This creates a simple plan–execute–adapt–replan loop: the model uses predictions to select actions, the actions lead to new observations from the environment, the new observations are used to update the model, and the updated model is immediately reused for the next planning step with better predictions. Our experiments show that one gradient step per replanning step, or a few when the prediction error is high, recovers substantial planning performance under various shifts, and is more sample efficient than offline fine-tuning. More broadly, latent world models should continue to be trained at deployment, rather than kept frozen. We believe this work opens a promising direction for adaptive world models that continuously calibrate predictions and update representations while acting, enabling more resilient perception and planning in a changing world.

AI USE STATEMENT

In this work, we used generative AI tools for polishing writing, implementing methods and supporting qualitative data analysis. We have not used generative AI tools for generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, assist with translation, clean and reformat dataset, interpret results. We have reviewed all AI-assisted work and LLM-generated code was verified and tested for correctness through multi-turn reviews. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. *ICML*, 2025.

Mido 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.

Randall Balestriero and Yann LeCun. Lejepa: Provable and scalable self-supervised learning without the heuristics. *arXiv preprint arXiv:2511.08544*, 2025.

Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. *ICLR*, 2022.

Amy J Bastian. Learning to predict the future: the cerebellum adapts feedforward movement control. *Current opinion in neurobiology*, 2006.

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. *CVPR*, 2021.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *IJRR*, 2025.

Kenneth J. W. Craik. *The Nature of Explanation*. Cambridge University Press, 1943.

Nathaniel D Daw, Samuel J Gershman, Ben Seymour, Peter Dayan, and Raymond J Dolan. Model-based influences on humans' choices and striatal prediction errors. *Neuron*, 2011.

J. Kahan Foigel and J. Richalet. Self-adapting idcom. *IFAC Proceedings Volumes*, 1979.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *ICLR*, 2021.

Yossi Gandelsman, Yu Sun, Xinlei Chen, and Alexei Efros. Test-time training with masked autoencoders. *NeurIPS*, 2022.

Shenyuan Gao, Siyuan Zhou, Yilun Du, Jun Zhang, and Chuang Gan. Adaworld: Learning adaptable world models with latent actions. *ICML*, 2025.

Carlos E. García, David M. Prett, and Manfred Morari. Model predictive control: Theory and practicea survey. *Automatica*, 1989.

Jan Gläscher, Nathaniel D. Daw, Peter Dayan, and John P. ODoherty. States versus rewards: Dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron*, 2010.

Raktim Gautam Goswami, Amir Bar, David Fan, Tsung-Yen Yang, Gaoyue Zhou, Prashanth Krishnamurthy, Michael Rabbat, Farshad Khorrami, and Yann LeCun. World models can leverage human videos for dexterous manipulation. *arXiv preprint arXiv:2512.13644*, 2025.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *ICML*, 2022.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *ICLR*, 2024.

Yining Hong, Huang Huang, Manling Li, Li Fei-Fei, Jiajun Wu, and Yejin Choi. Learning from trials and errors: Reflective test-time planning for embodied llms. *arXiv preprint arXiv:2602.21198*, 2026.

Basil Kouvaritakis and Mark Cannon. Model predictive control: Classical, robust and stochastic. 2016.

JB Lanier, Kyungmin Kim, Armin Karamzade, Yifei Liu, Ankita Sinha, Kat He, Davide Corsi, and Roy Fox. Adapting world models with latent-state dynamics residuals. *arXiv preprint arXiv:2504.02252*, 2025.

Yann LeCun. A path towards autonomous machine intelligence. 2022. URL `https://openreview.net/pdf?id=BZ5a1r-kVsf`.

Calvin Luo, Zilai Zeng, Mingxi Jia, Yilun Du, and Chen Sun. Self-improving loops for visual robotic planning. 2026.

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

Matthew R Nassar, Robert C Wilson, Benjamin Heasly, and Joshua I Gold. An approximately bayesian delta-rule model explains the dynamics of belief updating in a changing environment. *Journal of Neuroscience*, 2010.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. *ICML*, 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. *arXiv preprint arXiv:2302.12400*, 2023.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. Ogbench: Benchmarking offline goal-conditioned rl. In *International Conference on Learning Representations*, 2025.

Arjun Parthasarathy, Nimit Kalra, Rohun Agrawal, Yann LeCun, Oumayma Bounou, Pavel Izmailov, and Micah Goldblum. Closing the train-test gap in world models for gradient-based planning. *arXiv preprint arXiv:2512.09929*, 2025.

Reuven Y Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 1997.

Reza Shadmehr and F. A. Mussa-Ivaldi. Adaptive representation of dynamics during learning of a motor task. *Journal of neuroscience*, 1994.

Reza Shadmehr, Maurice A Smith, and John W Krakauer. Error correction, sensory prediction, and adaptation in motor control. *Annual review of neuroscience*, 2010.

Vlad Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim G. J. Rudner, and Yann LeCun. Learning from reward-free offline data: A case for planning with latent dynamics models. *NeurIPS*, 2025.

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

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 1991.

Sebastian Thrun, Knut Möller, and Alexander Linden. Planning with an adaptive world model. *NeurIPS*, 1990.

Leonardo F Toso, Davit Shadunts, Yunyang Lu, Nihal Sharma, Donglin Zhan, Nam H Nguyen, and James Anderson. Learning invariant visual representations for planning with joint-embedding predictive world models. *arXiv preprint arXiv:2602.18639*, 2026.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *ICLR*, 2021.

Hang Wang, Xin Ye, Feng Tao, Chenbin Pan, Abhirup Mallik, Burhaneddin Yaman, Liu Ren, and Junshan Zhang. Adawm: Adaptive world model based planning for autonomous driving. *ICLR*, 2025a.

Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. Continual test-time domain adaptation. *CVPR*, 2022.

Renhao Wang, Yu Sun, Arnuv Tandon, Yossi Gandelsman, Xinlei Chen, Alexei A Efros, and Xiaolong Wang. Test-time training on video streams. *JMLR*, 2025b.

Ying Wang, Oumayma Bounou, Gaoyue Zhou, Randall Balestriero, Tim GJ Rudner, Yann LeCun, and Mengye Ren. Temporal straightening for latent planning. *ICML*, 2026.

Daniel M Wolpert, R Chris Miall, and Mitsuo Kawato. Internal models in the cerebellum. *Trends in cognitive sciences*, 1998.

Hee Suk Yoon, Eunseop Yoon, Joshua Tian Jin Tee, Mark Hasegawa-Johnson, Yingzhen Li, and Chang D Yoo. C-tpt: Calibrated test-time prompt tuning for vision-language models via text feature dispersion. *ICLR*, 2024.

Marvin Zhang, Sergey Levine, and Chelsea Finn. Memo: Test time robustness via adaptation and augmentation. *NeurIPS*, 2022.

Wancong Zhang, Basile Terver, Artem Zholus, Soham Chitnis, Harsh Sutaria, Mido Assran, Randall Balestriero, Amir Bar, Adrien Bardes, Yann LeCun, and Nicolas Ballas. Hierarchical planning with latent world models. *arXiv preprint arXiv:2604.03208*, 2026.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. *ICML*, 2025.

A Environments and Data

A.1 PushT (DINO-WM)

PushT is a contact-rich manipulation environment introduced by (Chi et al., 2025). It consists of a circular pusher agent interacting with a T-shaped block. Starting from a random initial state, the agent must push the block to a target pose. Here, we use the setup from DINO-WM (Zhou et al., 2025), which treats the fixed green T as a visual reference rather than the target object itself. We use DINO-WM's PushT validation trajectories to compare frozen JEPA world models and their adaptive counterparts in Table 2.

A.2 PushObj

PushObj extends PushT by replacing the T-shaped block with objects of different shapes, {`L`, `Z`, `+`, `I`, `smallT`, `square`}, following (Zhou et al., 2025). However, simply replacing the object while replaying the original action sequences yields trajectories with much sparser contact: actions collected for the T-shaped block often no longer align with the new geometry, causing the agent to miss the object.

To construct PushObj, we start from $N = 18{,}500$ PushT training trajectories and introduce a contact bias to increase the fraction of trajectories in which the pusher interacts with the object. We generate $16{,}000$ training trajectories per shape. For testing, we filter the remaining trajectories to keep only those containing at least one contact. This makes the evaluation meaningful by filtering out trivial no-contact motion. We denote this dataset as `PushObj-all-16k` and have the following subsets for major experiments:

- `PushObj-TLZ+-4k`: we train on four shapes {`T`, `L`, `Z`, `+`} (each with 4k trajectories), and evaluate on the test trajectories of all seven shapes. This data is used in the **Shape Shifts** experiments shown in the upper row of Figure 3. We train the model for 3 epochs.
- `PushObj-T-16k`: we train on shape T (16k trajectories) only, and evaluate on various visual corruptions of its test trajectories. This data is used in the **Visual Shifts** experiments shown in the lower row of Figure 3. We train for 3 epochs.

A.3 PointMaze-Medium

PointMaze is a 2D navigation environment based on the MuJoCo physics engine (Fu et al., 2021). The task is to navigate from a start position to a target position, specified by start and goal images. The action space consists of forces applied along the x and y axes. We use Medium-Maze data from Wang et al. (2026) and directly use their pretrained checkpoint of ResNet global features for the **Dynamics Shifts** experiments as shown in Table 1. Unlike prior works that simply sample goal positions that are T steps away from the start in the test trajectories, we impose nontrivial goals in this environment. For PointMaze-Medium, there are 26 open grid cells of the medium maze. We start by picking two distinct cells without replacement, and then reject and resample until their Euclidean distance is larger than 3 cell units.

A.4 Diverse PointMaze

We largely follow the data-generation procedure of HWM-PLDM (Zhang et al., 2026) to generate 30 PointMaze environments with random 8×8 maze layouts where 25 are used for training (2,000 trajectories each, 50,000 in total) and 5 held-out layouts are saved for evaluation. At test time, we evaluate on the held-out layouts with 50 episodes in total, 10 per layout. For each episode, the start cell is sampled uniformly from the open cells, and the goal cell is sampled at a controlled shortest-path distance of 3–5 cells (easy) or 6–9 cells (hard) computed by BFS. This is used for the **Layout Shifts** experiments in Table 1. We train for 3 epochs.

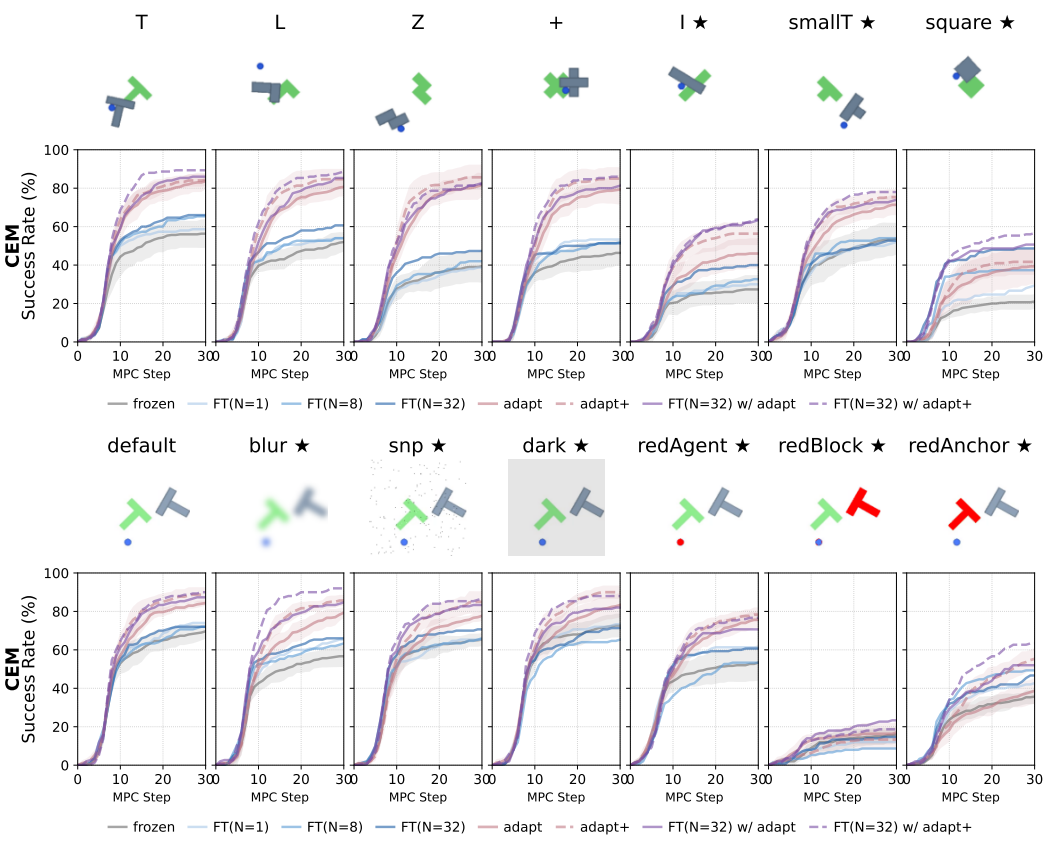


Figure 7: Planning success under shape shifts (top) and visual shifts (bottom) with the CEM sub-planner; otherwise as Figure 3.

B EXPERIMENTS

B.1 HYPERPARAMETERS

Table 3: Training Hyperparameters.

Name	Value
Encoder lr	1e-5
Predictor lr	5e-4
Action/Prop encoder lr	5e-4
Batch size	64
History frames	3
Frameskip	5

Table 4: Planning Hyperparameters.

Name	Value
Subplanner horizon	25
# Executed actions	5
GD Optimizer	Adam
GD Action Initialization	Zero
GD Learning rate	0.1
GD #opt steps	100
CEM # samples	200
CEM #opt steps	10

B.2 MPC WITH CEM

Figure 7 repeats the shape- and visual-shift comparison of Figure 3 with the CEM sub-planner. The ordering of the arms is the same as under GD: Adapt and Adapt+ are well above the frozen model on every shift the representation still covers, one-epoch fine-tuning on target-domain trajectories stays near frozen on all shifts but the square, and adaptation on top of the fine-tuned model is the best line on the unseen shapes and the sensor shifts.

B.3 The Red-Block Shift: What Fine-Tuning Restores and Adaptation Adds

Recoloring the block is the one shift on which online adaptation does not help (Figure 3). A ridge probe from the frozen visual latent to the true state explains why: on red-block frames the block's position is still decoded as well as under default colors (R^2 .96), but its orientation is not (.50 against .96), and PushT success needs the orientation. The orientation is lost inside the encoder's convolutional backbone, before the parameters adaptation updates, so neither the online step nor an offline fine-tune of the same parameters brings it back (probe R^2 .44 to .48). Fine-tuning the whole model on 32 target-domain trajectories for four epochs does (.92), and AdaJEPA applied on top of that model adds its usual gain (Table 5).

Table 5: Red-block shift, planning success. Fine-tuning uses $N{=}32$ target-domain trajectories

.

	GD	CEM
Frozen	15.0 ± 6.6	16.0 ± 3.8
Adapt+	17.0 ± 4.9	13.3 ± 5.0
FT last layers, 1 epoch	14.0 ± 2.8	14.7 ± 6.6
FT full, 4 epochs	52.0 ± 5.9	52.0 ± 4.3
FT full, 4 epochs w/ Adapt	71.3 ± 0.9	72.7 ± 4.1
FT full, 4 epochs w/ Adapt+	**76.7** ± 5.2	**83.3** ± 0.9

B.4 Ablations

We ablate the main design choices in AdaJEPA's test-time adaptation, including which parameters are updated, the adaptation learning rate, the number of gradient steps, and the replay-buffer design.

Setup. The predictor is a ViT-style module, consisting of a stack of transformer blocks followed by a final LayerNorm. The encoder is a small ResNet whose stages, in order, are five residual blocks (`rb1`–`rb5`), an optional pooling head, and a projection head. We define the encoder's *first stage* as `rb1`, the input residual block mapping $3 \to 32$ channels and extracting low-level features. We define its *last stage* as the projection head, a Linear–GELU–Linear–LayerNorm module that maps pooled features to the latent embedding. Unless otherwise specified, every adapting variant uses the same protocol: one gradient step per MPC replanning step, a replay buffer containing the five most recent transitions, a predictor learning rate of 5×10^{-4}, and an encoder learning rate of 10^{-5}. We use a maximum of 20 MPC steps(instead of 30 in the main experiments) to speed up experiments.

What to adapt. In general, AdaJEPA is robust to the choice of adaptation target. As shown in Figure 8, all adaptation variants improve over the frozen model across shape, visual, dynamics, and layout shifts. Updating only a small subset of parameters is often sufficient: `predlast+enclast` is consistently competitive, while adapting earlier predictor layers can further help under layout shifts. LoRA also improves over the frozen model but does not consistently outperform direct adaptation of selected predictor and encoder layers. The different variants are described below.

- `Frozen`. No test-time adaptation. The encoder and predictor are kept fixed throughout planning.
- `predlast+enclast`. Adapts the predictor's *last* transformer block, including the final LayerNorm, and the encoder's *last* stage.
- `predfirst+enclast`. Adapts the predictor's *first* transformer block and the encoder's *last* stage.
- `predfirstlast+enclast`. Adapts the predictor's *first* and *last* transformer blocks, including the final LayerNorm, and the encoder's *last* stage.
- `predlast+encfirst`. Adapts the predictor's *last* transformer block, including the final LayerNorm, and the encoder's *first* stage.
- `predlast+encfrozen`. Adapts only the predictor's *last* transformer block, including the final LayerNorm. The encoder remains frozen.
- `LoRA`. Inserts low-rank adapters with rank 8 and $\alpha = 16$ into every linear layer of the full predictor and encoder. Only the adapter parameters are updated, while all base weights remain frozen.

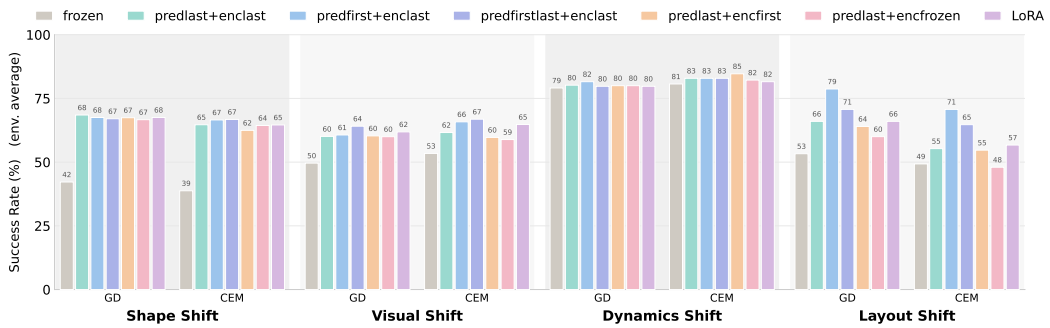
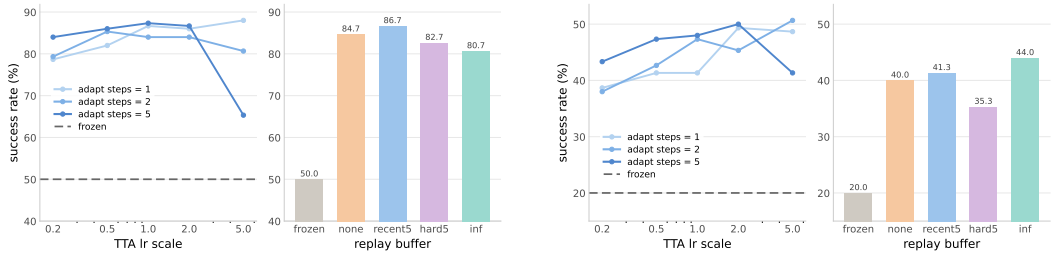


Figure 8: Effect of Adaptation Layers to Planning Success Rates. The reported values are the per-shift success rates (%) averaged over all setups within each shift. Test-time adaptation improves planning across all distribution shifts and is largely insensitive to which layers are adapted.



(a) T (seen shape)

(b) Square (unseen shape)

Figure 9: Effect of Adaptation Hyperparameters and Replay Buffers to Planning Success Rates.

Learning rates, optimization steps, and replay buffers. We ablate test-time adaptation hyperparameters and replay-buffer design on PushObj using a seen shape, `T`, and a held-out shape, `square` (Figure 9). Across the grid, adaptation is consistently effective: every setting substantially outperforms the frozen world model, improving success from 50% to 88% on `T`, and from 20% to 51% on `square`.

The learning rate and the number of gradient steps are tightly coupled. A large learning rate ($5\times$ the training rate) is highly effective with a single step, but can overshoot when combined with multiple steps. Conversely, a small learning rate ($0.2\times$) is more stable but requires more updates to become competitive, increasing latency. Moderate settings (1–$2\times$, one or two steps) are robust across both shapes, although the optimum remains environment- and model-dependent. To keep adaptation simple and efficient, we use the training learning rate and take one gradient step per MPC replanning step by default, which is competitive on both shapes at minimal cost. We further explore Adapt+, which lets the prediction loss decide the number of steps, to boost performance.

Replay-buffer design has a smaller effect: success varies only moderately across buffer choices, ranging from 81%–87% on `T` and 35%–44% on `square`. Importantly, every replay-buffer variant, including no buffer, substantially outperforms the frozen model. We use a recent sliding-window buffer in the main experiments, as it provides the most stable gains.

B.5 Additional Experiments: Training Data Scale for PushObj

We study how training-data size and diversity affect test-time adaptation on PushObj, which contains seven object shapes. We vary the number of training shapes, $K \in \{1, 2, 4\}$, and the number of trajectories per shape, $N \in \{1\text{k}, 2\text{k}, 4\text{k}, 8\text{k}, 16\text{k}\}$. For each K, we construct seven balanced training sets, each containing K shapes, so that every shape appears in exactly K sets. We train one world model per set using $K \times N$ trajectories, keeping the number of optimization steps fixed across settings to isolate the effect of data. At test time, we evaluate each model on all seven shapes, label each shape as *seen* or *unseen* according to the corresponding training set, and report success averaged

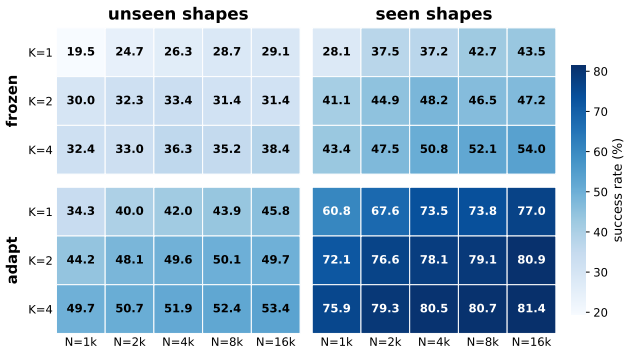


Figure 10: Heatmap of offline training data scale vs. PushObj planning success. Rows vary shape diversity K and columns vary trajectories per shape N.

over three seeds and over shapes in each group. We plot the results using line charts in Figure 6 and heatmaps in Figure 10.

- **Both more trajectories and more shape diversity are beneficial**: the frozen model improves with both more trajectories and more shape diversity: from (K=1, N=1k) to (K=4, N=16k), success increases from 28% to 54% on seen shapes and from 20% to 38% on unseen shapes.
- **Diversity matters more than the number of trajectories per shape**: with 16k trajectories total, distributing them over four shapes (K=4, N=4k) outperforms concentrating them on a single shape (K=1, N=16k) on both seen (81% vs. 77% adapted, 51% vs. 44% frozen) and unseen shapes (52% vs. 46% adapted, 36% vs. 29% frozen).
- **AdaJEPA provides a consistent gain at every data scale and can even compensate for large reductions in training data**: test-time adaptation leads to an average improvement of over 30% on seen shapes and over 15% on unseen shapes. Notably, adaptation can offset large reductions in training data. On seen shapes, the adapted model trained on a single shape with only 1k trajectories reaches 61%, exceeding the largest frozen model trained on four shapes with 16k trajectories each (64k total, 54%). On unseen shapes, an adapted single-shape model already surpasses the best four-shape frozen model once $N \geq$ 2k.

In general, test-time adaptation is most valuable when data is scarce, yet its benefits persist as data scales, providing a sample-efficient path to robust generalization.

B.6 Additional Experiments: OGBench Cube

We also evaluate on the single-cube manipulation task of OGBench (Park et al., 2025), following the cube setup of LeWorldModel (Maes et al., 2026): a JEPA world model trained on the cube dataset, planning tasks defined as 25-step segments of held-out expert episodes with the segment's last frame as the goal, and success judged on the cube's final position. Many segments barely move the cube, so a task is discarded when the cube's start and goal positions are within 0.05 m, which removes about 42% of the segments. We evaluate 50 tasks per seed over three seeds with 30 replanning steps and the CEM sub-planner (200 samples, 10 iterations). AdaJEPA consistently improves the performance on this task (Table 6).

Table 6: OGBench cube, in distribution, planning success.

Frozen	56.7 $\pm$ 5.2
Adapt	60.7 $\pm$ 3.4 $\uparrow$4.0
Adapt+	64.0 $\pm$ 1.6 $\uparrow$7.3

C VISUALIZATION OF PLANNING TRAJECTORIES

Here, we provide a qualitative comparison of single planning episodes, comparing the frozen model and AdaJEPA under identical start/goal conditions.

For each example, the upper rows show the trajectory executed for each method. The *simulator* row shows the rendered environment, while the *decoder* row shows the model's imagined latent rollout decoded to pixels. We show the frozen run for 13 steps (capped for better visibility), and gray out AdaJEPA steps after it reaches the goal. A check mark or cross indicates planning success or failure. The bottom panel reports the latent prediction loss at each replanning step. Before the current adaptation update, the model predicts a short latent rollout; we compare these predictions to the latents encoded from the observations actually reached by executing the planned actions. AdaJEPA reduces this loss by continuously refining the model's predictions, whereas the frozen model often fails due to inaccurate predictions.

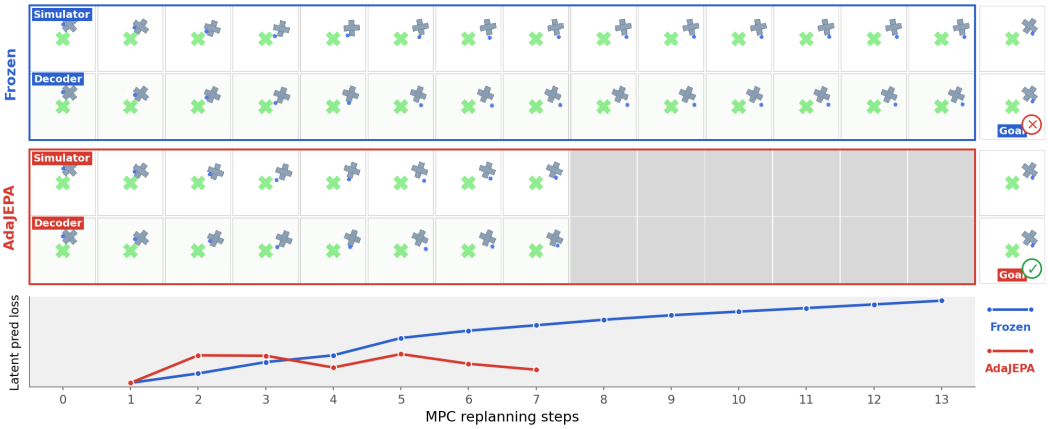


Figure 11: The model is trained on shape {T,L,+,Z}, and tested on a seen shape + here.

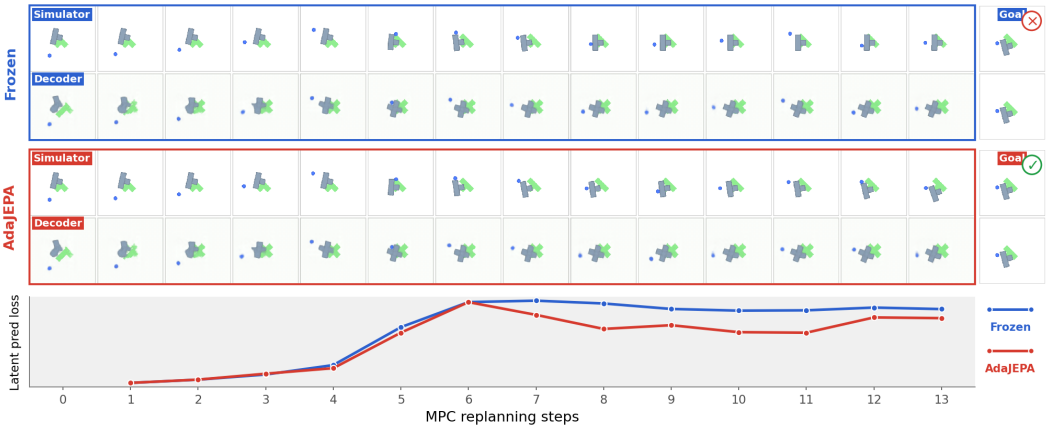


Figure 12: Shape Shifts: The model is trained on shapes {`T`, `L`, `Z`, `+`}, and tested on an unseen shape `smallT`. The decoder tries to decode to the closest seen shapes.

When facing visual corruptions and color changes at test time, the representation and prediction also need adaptation. AdaJEPA reduces the prediction loss consistently and leads to better planning. Since the decoder is only trained on default PushT data, it tends to reconstruct the original colors and scenes.

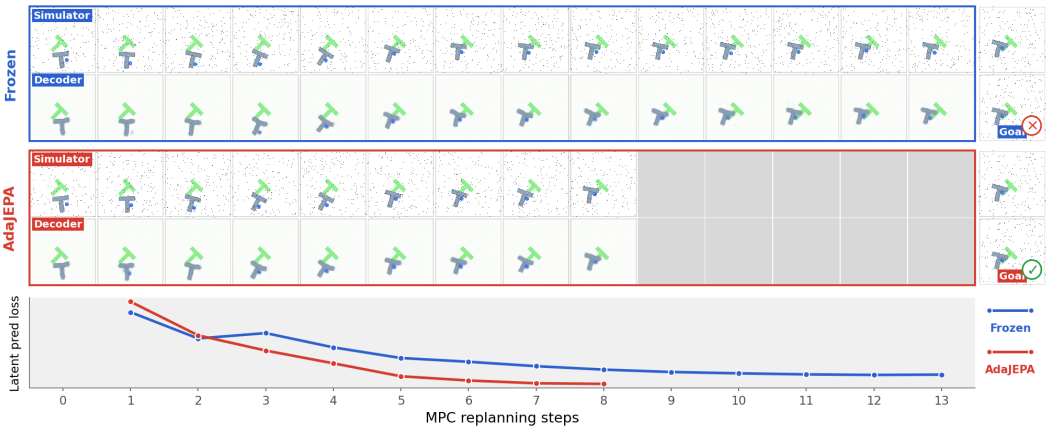


Figure 13: Visual Shifts: The model is trained on the default PushT data, but the test-time observations have salt-and-pepper noise. AdaJEPA consistently decreases prediction loss.

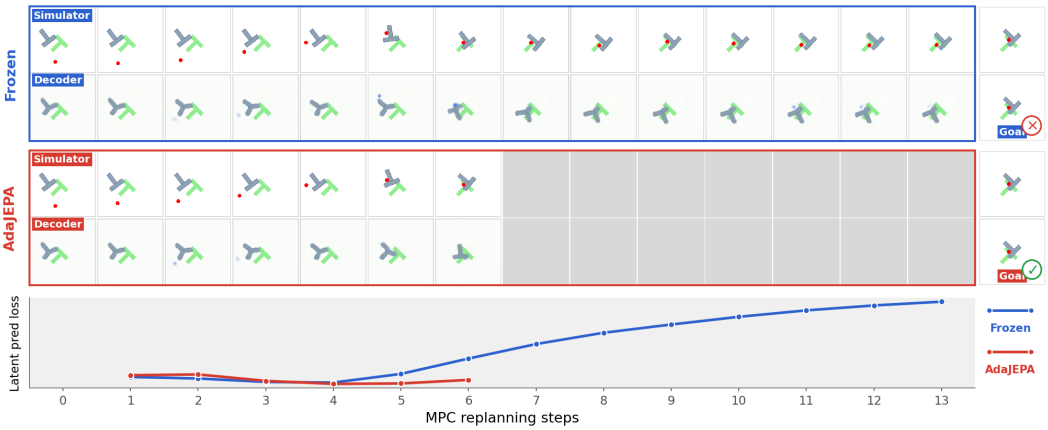


Figure 14: Visual Shifts: The model is trained on the default PushT data (agent is blue), but the agent is red at test time. AdaJEPA consistently decreases prediction loss.