# RECURSIVELY SELF-IMPROVING WORLD MODELS FOR REINFORCEMENT LEARNING

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

## ABSTRACT

Model-based reinforcement learning (MBRL) achieves strong data efficiency by leveraging synthetic experience for policy optimization; however, training a world model still requires substantial real interaction data, not only for accurate predictions but also for adequate coverage of real transitions. In this work, we investigate whether a world model can improve itself through its own generated experience to better support policy learning. We show that self-training amplifies concentration tendencies in generated experience, arising from mode-seeking sampling in unconditional generators and coupled model–policy interaction in conditional dynamics models. Reversing the resulting parameter displacement through negative extrapolation steers the world model toward generating more diverse and useful synthetic experience for downstream policy learning. Across both unconditional and conditional world-model formulations, our approach improves policy performance and data efficiency over the respective backbones on MuJoCo, DeepMind Control Suite, and Meta-World tasks with modest additional computation.

## 1 INTRODUCTION

Model-based reinforcement learning (MBRL) achieves strong data efficiency by leveraging synthetic experience for policy optimization (Janner et al., 2019; Hafner et al., 2025). While effective, this paradigm still requires substantial real interaction data to train the world model, not only to achieve accurate predictions (Yu et al., 2020b; Eysenbach et al., 2022) but also to ensure that model-generated experience adequately represents the diversity of the real transition distribution (Nguyen-Tang & Arora, 2023). Since model-generated data can improve the policy, a natural question arises: *can such data also be used to recursively improve the world model itself?*

In image generation, the low cost of synthesizing data has motivated methods that leverage model-generated samples to supplement scarce real-world datasets (Bertrand et al., 2024; Alemohammad et al., 2024). However, naively training models on their own synthetic outputs can reinforce sampling biases over successive generations (Bertrand et al., 2024; Gerstgrasser et al., 2024), ultimately resulting in model collapse (Gerstgrasser et al., 2024; Shumailov et al., 2024). Recent work further shows that the parameter displacement induced by such self-training can contain useful information beyond the degradation caused by recursive training (Alemohammad et al., 2025).

From a generative perspective, a world model is also a data generator, but its synthetic experience ultimately serves policy learning. We observe that this experience tends to concentrate through mode-seeking sampling in unconditional generators and recursive model–policy interaction in conditional dynamics models. Recursive self-training further amplifies this concentration, inducing a corresponding parameter displacement, while reversing this displacement steers the world model toward generating experience with better coverage.

Motivated by this insight, we propose a self-improving approach that exploits self-training-induced parameter displacement as an informative direction for recursively improving the world model. As illustrated in Figure 1, we instantiate this approach across both unconditional and conditional world-model formulations, showing how concentration emerges through distinct mechanisms and how our approach mitigates this tendency, ultimately improving policy performance and data efficiency.

Our main contributions are:



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.

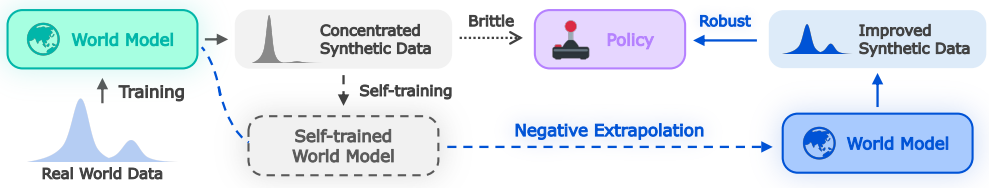



Figure 1: Self-training-induced parameter displacement provides a general self-improvement signal for world models. Reversing this displacement mitigates concentration in generated experience and yields more useful synthetic data for policy learning.

- We reveal that self-training-induced parameter displacement provides a general signal for world-model self-improvement, and characterize how concentration emerges through different mechanisms in unconditional and conditional settings.
- We instantiate this principle in two complementary world-model settings, introducing $S^2$ER for unconditional generation and REDREAM for conditional latent dynamics, both with modest computational overhead.
- Across diverse control tasks in MuJoCo, DMC, and Meta-World, both methods consistently improve policy performance and data efficiency over their respective backbones, demonstrating the effectiveness and generality of the proposed self-improvement principle.

## 2 Related Work

**Model-based reinforcement learning.** Model-based reinforcement learning (MBRL) leverages a predictive environment model, or *world model*, to improve decision making (Feinberg et al., 2018; Hafner et al., 2019b; Ye et al., 2021; Micheli et al., 2024; 2022). From a generative-modeling perspective, existing approaches can be broadly grouped into direct transition or trajectory generation and recursive generation through world-model–policy interaction. Diffuser (Janner et al., 2022) generates state–action trajectories for offline planning, while Decision Diffuser (Ajay et al., 2022) generates state trajectories and recovers actions through inverse dynamics. For policy learning, MT-Diff (He et al., 2023) synthesizes multi-task trajectories, SYNTHER (Lu et al., 2023) models the joint distribution of collected transitions to augment replay, and PGR (Wang et al., 2025) prioritizes informative experience through curiosity- or value-based conditioning.

Conditional dynamics models instead generate imagined trajectories by alternating policy action selection and model prediction. Representative approaches include probabilistic ensembles such as PETS (Chua et al., 2018), feed-forward models such as MBPO (Janner et al., 2019), recurrent latent models such as Dreamer (Hafner et al., 2019a; 2023), and Transformer-based world models such as STORM (Zhang et al., 2023); ADMPO (Lin et al., 2025) improves long-horizon prediction through any-step dynamics. In contrast, we use generated experience to recursively improve the world model in both settings and study how concentration induced by recursive model–policy interaction provides a useful direction for latent-model self-improvement.

**Model collapse and self-improving models.** Recursive training on a model's own outputs can amplify sampling biases and approximation errors, degrading generation quality and diversity and eventually causing model collapse (Shumailov et al., 2024; Alemohammad et al., 2024). Subsequent work studies the stability of recursive training (Bertrand et al., 2024) and mechanisms for preventing degradation. Self-Correcting Self-Consuming Loops (Gillman et al., 2024) uses correction functions to stabilize training on mixtures of real and recursively generated data, whereas NEON (Alemohammad et al., 2025) reverses the self-training-induced parameter displacement through negative extrapolation to improve image generation. More broadly, self-improvement in language models has been explored through bootstrapping from self-generated reasoning (Zelikman et al., 2022), assigning rewards to their own outputs (Yuan et al., 2024), or exploiting stronger internal verification than generation to construct training signals (Huang et al., 2025). In MBRL, generated experience has traditionally been used to improve planning (Hafner et al., 2019b), value estimation (Oh et al., 2017), or policy learning (Janner et al., 2019), rather than the world model itself. Recent work has begun to explore self-improving world models through forward–inverse verification (Liu et al.,

2026) or latent-action learning (Qiu et al., 2026). In contrast, we use the displacement induced by a world model's own generated experience as the improvement signal, without introducing a separate learned verifier or inverse model.

## 3 BACKGROUND

**Model-based reinforcement learning.** We consider a Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \rho_0, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, $P(\boldsymbol{s}' \mid \boldsymbol{s}, \boldsymbol{a})$ the environment dynamics, $R(r \mid \boldsymbol{s}, \boldsymbol{a})$ the reward distribution, $\rho_0$ the initial-state distribution, and $\gamma \in [0, 1)$ the discount factor. A policy $\pi_\theta(\boldsymbol{a} \mid \boldsymbol{s})$ seeks to maximize

$$\theta^\star = \arg\max_\theta \mathbb{E}_{\tau \sim p_{\mathrm{env},\theta}} \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right], \tag{1}$$

where $\tau = (\boldsymbol{s}_t, \boldsymbol{a}_t, r_t)_{t \geq 0}$ and $p_{\mathrm{env},\theta}$ is the trajectory distribution induced by $\rho_0$, $\pi_\theta$, $P$, and $R$. MBRL improves interaction efficiency by learning a world model $\mathcal{T}_\phi$ from real transitions in the replay buffer $\mathcal{D}_{\mathrm{real}}$ and using model-generated experience for policy learning. The form of synthetic experience depends on the world-model formulation. Unconditional generative models learn the joint replay distribution and directly sample complete transitions $\widehat{\xi} = (\widehat{\boldsymbol{s}}, \widehat{\boldsymbol{a}}, \widehat{\boldsymbol{s}}', \widehat{r}) \sim q_\phi$, where the action is generated jointly with the other transition variables rather than selected by the current policy. Conditional dynamics models instead generate imagined experience through recursive interaction with the policy: $p_{\phi,\theta}(\widehat{\tau} \mid \boldsymbol{s}_0) = \prod_{t=0}^{H-1} \pi_\theta(\boldsymbol{a}_t \mid \widehat{\boldsymbol{s}}_t)\, p_\phi\left(\widehat{\boldsymbol{s}}_{t+1}, \widehat{r}_t \mid \widehat{\boldsymbol{s}}_t, \boldsymbol{a}_t\right)$, where $\widehat{\boldsymbol{s}}_0 = \boldsymbol{s}_0$ and $\widehat{\tau}$ denotes an $H$-step imagined trajectory.

**Negative extrapolation from self-training.** Negative extrapolation from self-training (NEON) turns self-training-induced degradation into a corrective signal (Alemohammad et al., 2025). Given a base generator $G_\phi$ and sampling hyperparameters $\kappa$, NEON draws synthetic samples from the induced distribution $q_{\phi,\kappa}$ and briefly fine-tunes $G_\phi$ on them to obtain $\phi_{\mathrm{self}}$. Mode-seeking samplers favor high-density regions already well represented by the model, so self-training reinforces this sampling preference in the parameters, strengthening dominant modes and reducing coverage. NEON instead reverses the resulting displacement through

$$\phi_{\mathrm{NEON}} = (1 + w)\phi - w\phi_{\mathrm{self}}, \tag{2}$$

where $w > 0$ controls the negative-extrapolation strength. Sufficiently mild negative extrapolation can instead reduce risk by shifting probability mass from over-represented to under-represented modes, typically trading precision for improved recall and distributional fidelity.

## 4 RECURSIVELY SELF-IMPROVING WORLD MODELS

We call a world model *recursively self-improving* when its own generated experience can be used to derive a corrective update that improves the utility of its subsequent synthetic experience for policy learning. We first provide the intuition behind this self-improving principle, then instantiate it as S$^2$ER for unconditional generative replay and REDREAM for conditional latent dynamics. Finally, we examine why the self-improvement principle works in both settings.

### 4.1 MOTIVATION

World models in RL can be viewed as generative models of environment experience: they learn from real agent–environment interactions and synthesize data for planning and policy optimization. Broadly speaking, they fall into two categories: unconditional models sample complete transitions from a learned replay distribution, while conditional models predict future transitions given states and policy actions. Once prediction becomes sufficiently accurate, another question becomes central: *what distributional properties should synthetic experience have?*

Prior work highlights two properties important for policy learning: real–synthetic distributional fidelity (Lu et al., 2023) and mode coverage (Nguyen-Tang & Arora, 2023). Additional real interactions could improve both properties, but environment interaction is precisely the limited resource

that model-based RL aims to conserve. This motivates a different question: whether a world model can improve by reusing its own synthetic experience.

We observe that world models already exhibit a certain "preference" in the data they generate. In unconditional cases, this preference arises from mode-seeking sampling, whereas in conditional cases, it arises from recursive model–policy interaction. In both cases, naive self-training reinforces this preference, resulting in distributional contraction. Intuitively, this degradation provides a structured direction for recovery or, with a mild update in the opposite direction, can even steer the model toward generating data with better coverage, ultimately benefiting policy learning. We study negative extrapolation as a simple mechanism to realize this principle and instantiate it in both settings.

### 4.2 Self-Improving Unconditional Generative Models

In this section, we propose **Self-Improving Synthetic Experience Replay ($S^2$ER)** as an instantiation of our recursively self-improving principle for unconditional generative world models. We use Synthetic Experience Replay (SynthER) (Lu et al., 2023) as the generative backbone, which learns from collected transitions and generates synthetic experience unconditionally for policy training.

The core idea is that self-training reinforces high-density regions favored by diffusion samplers, progressively concentrating probability mass and reducing distributional coverage. Such misalignment is particularly problematic in MBRL because the policy is optimized using model-generated experience but ultimately evaluated in the real environment. It can therefore be exploited during policy optimization, yielding policies that perform well under the learned model yet poorly in the true environment (Janner et al., 2019; Yu et al., 2020b).

Reversing the resulting parameter displacement can instead redistribute probability mass toward under-represented modes, improving mode coverage and real–synthetic alignment (visualized in Appendix A). However, excessive extrapolation can also reduce transition fidelity, which may encourage the policy to exploit model errors and destabilize training (Eysenbach et al., 2022; Rigter et al., 2023). It is therefore natural to seek improved real–synthetic alignment while limiting degradation in transition fidelity. The detailed procedure is as follows:

**Degradation threshold.** We first constrain degradation in held-out real-data fidelity. Specifically, we evaluate each candidate diffusion model using its denoising loss on held-out real transitions and reject candidates whose loss exceeds a prescribed tolerance.

At each diffusion-model training event, we resample $\mathcal{D}_{\mathrm{val}}$ by drawing 1k real transitions from the current replay buffer and exclude them from the diffusion-model fitting and normalization-statistics estimation. All candidates within the event share this validation set. Let $z = \mathrm{norm}(x) \in \mathbb{R}^p$ denote the normalized representation of transition $x$. We define

$$L(\phi) = \mathbb{E}_{x\sim\mathcal{D}_{\mathrm{val}},\ \epsilon\sim\mathcal{N}(0,I),\ \sigma\sim p_\sigma}\left[\lambda(\sigma)\frac{\|D_\phi(z+\sigma\epsilon,\sigma)-z\|_2^2}{p}\right], \tag{3}$$

where $D_\phi$ denotes the denoiser, $\sigma_{\mathrm{data}}$ is a fixed data-scale hyperparameter for normalized clean data, and $\lambda(\sigma) = (\sigma^2+\sigma_{\mathrm{data}}^2)/(\sigma^2\sigma_{\mathrm{data}}^2)$. A candidate model $\phi'$ is accepted only if $L(\phi') \le (1+\gamma_{\mathrm{gate}})L(\phi)$, where $\phi$ denotes the reference model before self-training.

**Real–synthetic alignment.** We next use Transition Fréchet Distance (tFID) to quantify the distributional discrepancy between generated and held-out real transitions in the normalized representation space. Among candidates satisfying the held-out real-loss constraint, a lower tFID indicates better alignment with the real transition distribution. Let $(\boldsymbol{\mu}_{\mathrm{r}}, \boldsymbol{\Sigma}_{\mathrm{r}})$ and $(\boldsymbol{\mu}_\phi, \boldsymbol{\Sigma}_\phi)$ denote the empirical mean vectors and covariance matrices of normalized real and synthetic transitions. We define

$$\mathrm{tFID}(\phi) = \|\boldsymbol{\mu}_{\mathrm{r}} - \boldsymbol{\mu}_\phi\|_2^2 + \mathrm{Tr}\left(\boldsymbol{\Sigma}_{\mathrm{r}} + \boldsymbol{\Sigma}_\phi - 2\left(\boldsymbol{\Sigma}_{\mathrm{r}}^{1/2}\boldsymbol{\Sigma}_\phi\boldsymbol{\Sigma}_{\mathrm{r}}^{1/2}\right)^{1/2}\right). \tag{4}$$

Algorithm 1 summarizes the $S^2$ER procedure. Every $n$ environment steps, we refresh the diffusion model $\mathcal{T}_\phi$ on the accumulated real replay and immediately perform one self-improving procedure using its generated transitions. The final self-improved model is obtained through grid search: during self-training, we save checkpoints $\{\phi_{\mathrm{self}}^{(k)}\}$ corresponding to increasing self-training depths.

**Algorithm 1 S$^2$ER: Self-Improving SYNTHER**. Our additions are in blue.

**Input:** real-data ratio $r$; interval $n$, checkpoints $\mathcal{K}$, weights $\mathcal{W}$, and threshold $\gamma_{\text{gate}}$
**Initialize:** $\mathcal{D}_{\text{real}}, \mathcal{D}_{\text{val}}, \mathcal{D}_{\text{syn}} \leftarrow \varnothing$, agent $\pi$, and diffusion model $\mathcal{T}_\phi$
**for** $t = n, 2n, \ldots, N$ **do**
  Collect new real experience into $\mathcal{D}_{\text{real}}$ and resample a held-out $\mathcal{D}_{\text{val}}$
  Train $\mathcal{T}_\phi$ on the remaining real replay and generate $\mathcal{D}_{\text{syn}}$ from $\mathcal{T}_\phi$
  Self-train $\mathcal{T}_\phi$ on $\mathcal{D}_{\text{syn}}$ and periodically save $\{\phi_{\text{self}}^{(k)}\}_{k\in\mathcal{K}}$
  **for all** $(k, w) \in \mathcal{K} \times \mathcal{W}$ **do** ▷ grid search
    $\phi_{(k,w)} \leftarrow \phi - w\big(\phi_{\text{self}}^{(k)} - \phi\big)$ ▷ negative extrapolation
    Calculate $\ell_{(k,w)}$ and $d_{(k,w)}$ using Eqs. (3) and (4)
  Select the lowest-tFID candidate $\phi^*$ satisfying $\ell_{(k,w)} \le (1 + \gamma_{\text{gate}})L(\phi)$
  Regenerate $\mathcal{D}_{\text{syn}}$ using $\mathcal{T}_{\phi^*}$ and train $\pi$ on $\mathcal{D}_{\text{real}} \cup \mathcal{D}_{\text{syn}}$ mixed with real-data ratio $r$

We then construct negatively extrapolated candidates over checkpoint–weight pairs $(k, w)$ using Eq. (2) and evaluate them with the held-out real loss and tFID in Eqs. (3) and (4). Among candidates within the allowed real-loss increase over the reference model, we select the one with the lowest tFID to generate synthetic experience for policy training. For more details, Appendix A reports ablations on the refresh interval $n$ and gate threshold $\gamma_{\text{gate}}$, and further shows that reversing the self-training displacement yields a distributional improvement comparable to additional real-data fine-tuning. Appendix B presents a complete training instance and tracks the evolution of tFID, precision, and recall. Full hyperparameters are provided in Appendix E.

### 4.3 EXTENDING TO CONDITIONAL LATENT DYNAMICS MODELS

We extend the self-improving approach to conditional latent dynamics models using DreamerV3 (Hafner et al., 2023) as a representative example. Whereas SYNTHER directly samples complete transitions from a learned replay distribution, Dreamer generates future latent transitions sequentially through recursive interaction between the world model and the policy.

Our key insight is that this interaction can itself induce concentration. Self-training on this policy-conditioned distribution progressively reinforces frequently visited latent state–action regions, making subsequent imagined trajectories increasingly concentrated. This contraction reduces the coverage of imagined experience and can limit the diversity of the training signal available for subsequent policy learning. We therefore introduce **Recursively Self-Improving Dreamer (REDREAM)**, which applies negative extrapolation to reverse the resulting self-training displacement and counteract this contraction, as summarized in Algorithm 2. The detailed procedure is as follows:

**Latent synthetic data generation.** We adopt the recurrent state-space model (RSSM) used in DreamerV3, whose stochastic latent variables are categorical. We denote the world model by $\mathcal{T}_\phi$, with a composite latent state $\boldsymbol{s}_t = (\boldsymbol{h}_t, \boldsymbol{z}_t)$ consisting of a deterministic state $\boldsymbol{h}_t$ and a stochastic state $\boldsymbol{z}_t$. The latent state serves as the agent's memory, aggregating observations $\boldsymbol{o}_t$ and actions $\boldsymbol{a}_t$ to model the dynamics and predict rewards $r_t$ and continuation flags $c_t$.

For imagination, we infer posterior latent starting states from replayed sequences and roll out the current world model with the frozen policy. The complete rollout process is defined as

$$\begin{aligned}
&\textbf{Policy Model:} && \boldsymbol{a}_t \sim \pi_\theta(\boldsymbol{a}_t \mid \boldsymbol{s}_t),\\
&\textbf{Sequence Model:} && \boldsymbol{h}_{t+1} = f_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t),\\
&\textbf{Dynamics Predictor:} && \boldsymbol{z}_{t+1} \sim p_\phi(\boldsymbol{z}_{t+1} \mid \boldsymbol{h}_{t+1}),\\
&\textbf{Latent State:} && \boldsymbol{s}_{t+1} = (\boldsymbol{h}_{t+1}, \boldsymbol{z}_{t+1}),\\
&\textbf{Reward Predictor:} && \widehat{r}_{t+1} \sim p_\phi(\widehat{r}_{t+1} \mid \boldsymbol{s}_{t+1}),\\
&\textbf{Continuation Predictor:} && \widehat{c}_{t+1} \sim p_\phi(\widehat{c}_{t+1} \mid \boldsymbol{s}_{t+1}).
\end{aligned} \tag{5}$$

Each imagined trajectory has a rollout horizon of $H$, and we collect a total of $\eta$ imagined transitions to form the self-training dataset $\mathcal{D}_{\text{dream}}$. All generated transitions are detached before self-training and discarded after the self-improving update rather than stored in the real replay buffer.

**Latent self-training objective.** Unlike S$^2$ER in Section 4.2, we do not perform a grid search over self-training checkpoints. Instead, we apply a fixed negative-extrapolation correction with extrapo-

**Algorithm 2 REDREAM: Recursively Self-Improving Dreamer.** Our additions are in blue.

**Input:** interval $n$, rollout horizon $H$, self-training size $\eta$, and extrapolation weight $w$
**Initialize:** real replay buffer $\mathcal{D}_{\text{real}}$, Dreamer agent with world model $\mathcal{T}_\phi$ and policy $\pi$
**for** every $n$-th standard Dreamer update **do**
    Collect experience using $\pi$ and add it to $\mathcal{D}_{\text{real}}$
    Sample $\mathcal{B}_{\text{real}} \sim \mathcal{D}_{\text{real}}$ and perform a standard Dreamer update to obtain $\phi$
    Generate $\mathcal{D}_{\text{dream}}$ of size $\eta$ by rolling out $\mathcal{T}_\phi$ with the frozen policy for $H$ steps
    Self-train $\phi_{\text{rssm}}$ on $\mathcal{D}_{\text{dream}}$ using Eq. (6) and obtain $\phi^{\text{self}}_{\text{rssm}}$
    $\phi_{\text{rssm}} \leftarrow \phi_{\text{rssm}} - w(\phi^{\text{self}}_{\text{rssm}} - \phi_{\text{rssm}})$ ▷ negative extrapolation

lation weight $w$ to the RSSM after every $n$ Dreamer updates. The self-training objective is

$$\mathcal{L}_{\text{s}}(\phi) = \mathbb{E}_{\widehat{\xi}_t \sim \mathcal{D}_{\text{dream}}}\Big[ -\log p_\phi(\boldsymbol{z}_{t+1} \mid \boldsymbol{h}_{t+1}) - \log p_\phi(\widehat{r}_{t+1} \mid \boldsymbol{s}_{t+1}) - \log p_\phi(\widehat{c}_{t+1} \mid \boldsymbol{s}_{t+1})\Big]. \quad (6)$$

Here, $\widehat{\xi}_t$ contains detached synthetic targets generated by the current world model. During self-training, only the RSSM parameters are updated; all other components remain frozen, while reward and continuation losses backpropagate through their latent inputs to the RSSM.

Compared with the full Dreamer objective, our self-training loss omits reconstruction because imagination remains entirely in latent space and therefore provides no reconstruction gradient. Despite this, the latent dynamics, reward, and continuation objectives still define an effective negative-extrapolation direction that improves policy learning. In contrast, introducing decoder-generated pseudo-observations for reconstruction degrades performance, as shown in Appendix C. Ablations on the extrapolation weight $w$ and correction interval $n$, together with training details and computational cost, are shown in Appendix D. Full hyperparameters are reported in Appendix F.

## 5 EXPERIMENTS

In this section, we answer three main questions. (1) To what extent does our self-improving approach improve policy performance and data efficiency across the two world-model settings? (2) How does negative extrapolation improve mode coverage and real–synthetic alignment in generative replay? (3) How does self-training-induced concentration emerge in conditional latent dynamics, and can reversing the resulting displacement improve policy learning?

**Unconditional generative-model setting.** Following prior work on generative RL (Lu et al., 2023), we adopt the same online RL evaluation suite for direct comparison. Our evaluation covers continuous-control tasks from both the DeepMind Control Suite (DMC) (Tunyasuvunakool et al., 2020) and OpenAI Gym MuJoCo (Brockman et al., 2016). We compare against SYNTHER (Lu et al., 2023) and PGR (Wang et al., 2025) as the most closely related generative experience-replay methods, together with the model-based baseline DreamerV3 (Hafner et al., 2023) and the model-free SAC (Haarnoja et al., 2018). For a fair comparison, PGR, SYNTHER, and S$^2$ER use the same diffusion-model and policy architectures, as well as the same environment-step refresh interval $n$.

**Conditional latent-dynamics setting.** For conditional latent dynamics models, we evaluate a fixed subset of 12 challenging Meta-World tasks (Yu et al., 2020a), selected prior to running REDREAM. These tasks span diverse robotic manipulation settings requiring precise control and interaction with objects of varying sizes. We compare REDREAM with DreamerV3 (Hafner et al., 2023) and TD-MPC2 (Hansen et al., 2024) across all evaluated tasks. To isolate the effect of self-improvement, all other training settings are kept identical to those of the DreamerV3 baseline.

### 5.1 RESULTS

**S$^2$ER.** As summarized in Table 1, S$^2$ER consistently improves over SYNTHER across all six MuJoCo and DMC tasks. It also achieves competitive performance against the other baselines across the six tasks. These results suggest that controlled self-improvement can enhance the utility of synthetic experience generated by unconditional generative world models for subsequent policy learning.

Table 1: Comparison of S$^2$ER and baselines on MuJoCo and DMC tasks after 100k environment steps (Walker2d 200k). Results are reported as mean ± standard deviation over 5 seeds. All baseline results are obtained by rerunning the corresponding official implementations.

| Method | MuJoCo | | | DMC (100k) | | |
|---|---|---|---|---|---|---|
| | Walker2d | HalfCheetah | Hopper | Cheetah -Run | Reacher -Hard | Finger-Turn -Hard |
| SAC | 1937.7±445.3 | 5071.9±520.3 | 1447.2±905.6 | 327.7±49.7 | 369.3±196.0 | 193.3±150.7 |
| DreamerV3 | 2728.9±534.2 | 3160.8±2115.8 | 2199.0±295.1 | 219.0±47.9 | 795.4±99.0 | 334.3±233.9 |
| PGR | 4961.1±694.5 | 8806.1±302.6 | 3361.9±93.7 | 677.0±109.9 | **945.1±31.5** | 557.6±190.9 |
| SYNTHER | 4813.5±269.3 | 8994.2±582.2 | 3440.8±93.8 | 838.0±36.0 | 811.7±223.6 | 316.8±84.9 |
| **S$^2$ER (ours)** | **5905.5±90.5** | **10134.9±399.3** | **3588.6±45.8** | **874.4±11.6** | **928.4±51.3** | **599.4±39.7** |

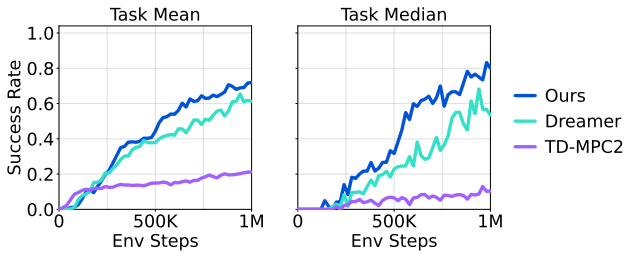



Figure 2: Comparison of REDREAM and baselines on 12 challenging Meta-World tasks across 3 seeds. We report the mean and median success rates across tasks. Details in Appendix G.

**REDREAM.** As shown in Figure 2, REDREAM improves the final mean success rate (SR) over DreamerV3, while increasing the median success rate from 0.53 to 0.80. These results indicate faster learning and improved data efficiency. Detailed per-task results are provided in Appendix G.

Importantly, these gains require only modest additional computation. Across the six-task MuJoCo and DMC evaluation, S$^2$ER increases total GPU-hours by only 1.7% over SYNTHER, while REDREAM incurs a 5.6% overhead over DreamerV3 across the full 12-task Meta-World evaluation. Taken together, the consistent improvements across both unconditional generative world models and conditional latent dynamics models demonstrate the effectiveness, computational efficiency, and generality of our self-improving approach.

### 5.2 UNDERLYING MECHANISM OF S$^2$ER'S GAIN

For S$^2$ER in the generative replay setting, we examine how self-training amplifies the generator's existing concentration tendency and how reversing the resulting displacement improves the reference model's mode coverage and real–synthetic alignment.

**Mode contraction and its reversal.** Following the same $k$-nearest-neighbor definition used by Alemohammad et al. (2025), we compute precision and recall with $k = 5$ using Euclidean distance in the same normalized transition space as tFID. For each sample, the neighborhood radius is defined by its fifth-nearest neighbor within the same set, excluding itself. Precision measures the fraction

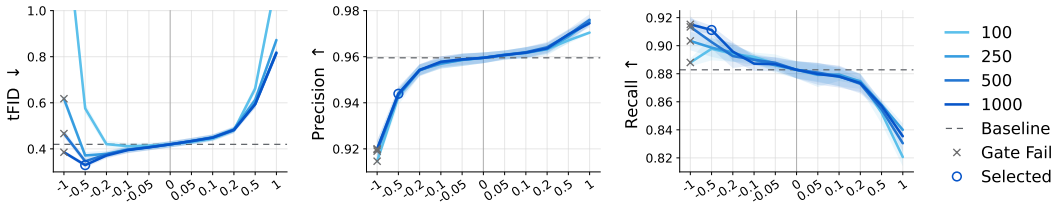
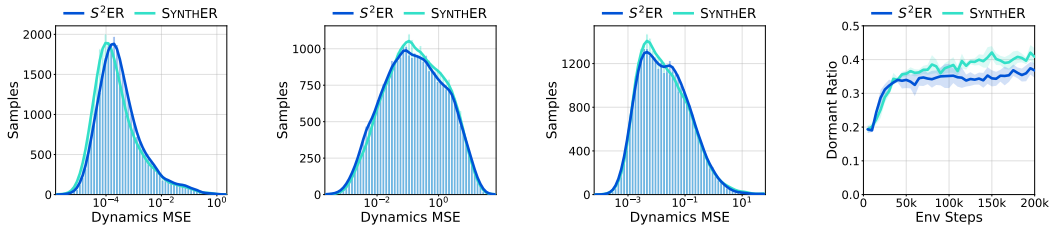



Figure 3: Mode contraction amplified by self-training and reversed by negative extrapolation on Walker2d at 50k steps. Colors denote self-training steps; negative displacement coefficients indicate negative extrapolation and positive coefficients indicate self-training.




(a) Hopper. (b) Walker2d. (c) HalfCheetah. (d) Dormant ratio.

Figure 4: Dynamics prediction error and dormant ratio of S$^2$ER and SYNTHER.

of synthetic transitions covered by real-data neighborhoods, whereas recall measures the fraction of real transitions covered by synthetic-data neighborhoods.

Figure 3 shows opposite trends along the self-training and negative-extrapolation directions. Moving along the self-training direction increases precision but progressively reduces recall and worsens tFID, consistent with concentration on a narrower subset of the real transition distribution. Moderate negative extrapolation reverses this trend: recall increases and tFID decreases, with only a modest reduction in precision. The gain in coverage is therefore accompanied by improved real–synthetic alignment, rather than simply producing broader but poorly matched samples.

However, overly strong negative extrapolation sharply reduces precision and worsens tFID; several such candidates are also rejected by the real-loss gate. Our constrained selection thus balances this trade-off, choosing a candidate that improves mode coverage and real–synthetic alignment while keeping the increase in held-out real loss within the prescribed tolerance.

**How does improved synthetic data help policy training?** We compare the dynamics MSEs of S$^2$ER and SYNTHER at 50k training steps using 10k synthetic transitions from each method. For each generated transition, we restore the simulator to its generated current state, execute the generated action, and compare the generated next state and reward with the corresponding ground-truth outcomes. As shown in Figure 4a–c, both methods have similar prediction error, with S$^2$ER showing only slightly higher dynamics MSE, consistent with the trade-off between prediction accuracy and improved distributional alignment and mode coverage. We further examine policy overfitting using the dormant ratio (DR) (Sokar et al., 2023), which has been associated with network overfitting (Xu et al., 2024). Figure 4d shows that S$^2$ER consistently maintains a lower DR, which is consistent with reduced policy overfitting under the broader synthetic-data distribution.

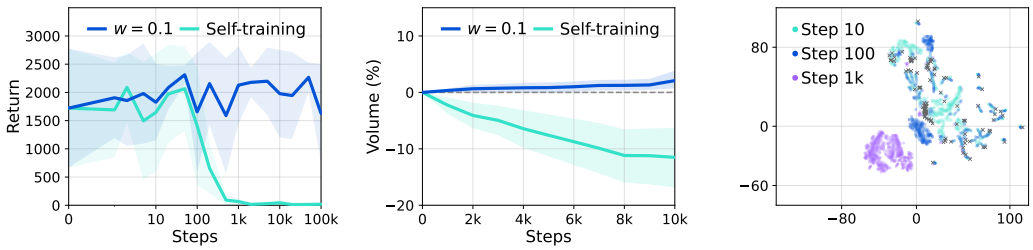
### 5.3 UNDERLYING MECHANISM OF REDREAM'S GAIN

We next use recursive self-training to expose the concentration tendency in policy-conditioned imagination. We then compare negative extrapolation with random and positive extrapolation controls to test whether reversing the displacement provides a useful correction during policy learning.

**Model collapse.** We first train DreamerV3 for $0.5$M environment steps on the Sweep task, take the resulting world model as the reference model $\phi$, and collect posterior latent states from the real replay buffer as a fixed pool of rollout starting states. Self-training then follows the standard Dreamer optimization settings, except that only the RSSM is updated using Eq. (6) on recursively generated imagined transitions, while the policy remains frozen. We then apply negative extrapolation to every saved self-training checkpoint with $w = 0.1$ and compare the evaluation return and action volume.

As shown in Figure 5a, the real environment evaluation return remains high during the early stages of self-training but subsequently drops sharply and eventually approaches zero, revealing a clear model-collapse pattern. This behavior is consistent with the collapse observed under recursive synthetic-data training (Dohmatob et al., 2024). In contrast, with mild negative extrapolation from the self-training displacement, the return remains close to its initial level.

**Coverage contraction.** We quantify how recursive self-training affects the dispersion of policy-conditioned imagined trajectories using the covariance volume of their final actions. Given 64 fixed rollout starting states, we generate 32 stochastic rollouts from each state with the frozen policy. For each starting state $s$, we estimate the final-action covariance $\Sigma_s$ and define its volume as $V_s = \sqrt{\det \Sigma_s}$; we report the geometric-mean volume change relative to the reference model.




(a) Evaluation return. (b) Action volume. (c) Latent distributions.

Figure 5: Model collapse and distribution contraction under recursive self-training.

As shown in Figure 5b, recursive self-training progressively contracts this distribution, reducing the covariance volume by approximately 11% after 10k updates. Negative extrapolation reverses this contraction and slightly expands the volume above the reference level. Since both the policy and rollout starting states are fixed, these changes reflect differences in the imagined trajectories induced by the evolving world model.

We further visualize the evolution of imagined trajectories in latent space using t-SNE, while keeping the rollout starting states fixed across self-training stages. As shown in Figure 5c, the imagined distribution initially overlaps closely with the real-data distribution (gray crosses), but progressively shifts and appears more concentrated under recursive self-training. Together with the action-space results, this pattern is consistent with a contraction in diversity under recursive self-training, while negative extrapolation reverses the associated concentration.

**Directionality matters.** To test whether the gains arise from the specific self-training displacement rather than perturbation magnitude, we compare negative extrapolation with an RMS-matched random control and positive extrapolation along the self-training direction. For each parameter tensor $\ell$, the random control replaces $\Delta\phi^{(\ell)}_{\text{true}} = \phi^{(\ell)}_{\text{self}} - \phi^{(\ell)}_{\text{ref}}$ with $\Delta\phi^{(\ell)}_{\text{rand}} = \sigma_\ell \boldsymbol{\epsilon}^{(\ell)}$, where $\sigma_\ell = \text{RMS}(\Delta\phi^{(\ell)}_{\text{true}})$ and $\boldsymbol{\epsilon}^{(\ell)} \sim \mathcal{N}(0, \boldsymbol{I})$. As shown in Figure 6, REDREAM with negative extrapolation learns substantially faster and achieves stronger performance on both Sweep and Push. Positive extrapolation is markedly less data-efficient, exhibiting only late recovery on Sweep and Push, while the random control consistently underperforms. These results show that the gain does not arise from an arbitrary same-scale perturbation or simply moving along the self-training direction, but specifically from reversing the structured self-training displacement.

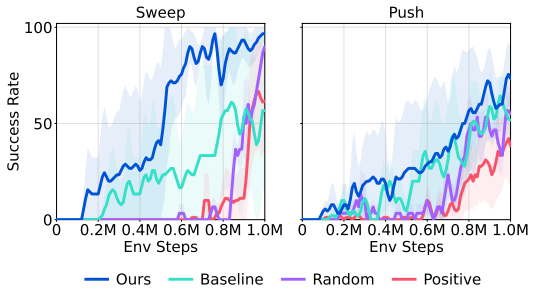



Figure 6: Directionality controls.

## 6 CONCLUSION

In this work, we show how self-training on a world model's generated experience can provide a direction for recursively improving the model itself. Self-training amplifies concentration tendencies in the generated distribution, and reversing the resulting parameter displacement steers the world model toward generating more useful synthetic experience for policy learning. This principle applies to sampler-induced concentration in unconditional generative replay and to model–policy interaction in conditional latent dynamics. Our two instantiations, S$^2$ER and REDREAM, improve policy performance and data efficiency over their respective backbones across MuJoCo, DMC, and Meta-World tasks with modest computational overhead.

More broadly, our findings suggest that the value of a world model is determined not only by its predictive fidelity, but also by how its generated experience shapes policy optimization. This perspective motivates a broader direction for MBRL in which world models and policies improve through a tighter feedback loop, with synthetic experience serving as an integral part of model improvement rather than only as additional data for policy training.

### AI USE STATEMENT

In this work, we used generative AI tools to assist with code modification, literature discovery, and language polishing of author-written text. We did not use generative AI tools to generate figures, experimental results, scientific claims, or substantive manuscript content. All experimental design, implementation decisions, analyses, results, and conclusions were produced and verified by the authors. All AI-assisted code modifications were manually reviewed and tested for correctness, and all language edits were reviewed to ensure that they faithfully preserved the authors' intended meaning. We take responsibility for the final content of this work, including any text or code modified with the assistance of generative AI.

### REPRODUCIBILITY STATEMENT

We provide the source code for all experiments in the supplementary materials. The main paper presents the algorithms and objectives for $S^2$ER and REDREAM. Detailed training procedures, ablations, and computational costs for $S^2$ER are provided in Appendix A, with a complete training instance in Appendix B. The reconstruction-loss ablation for REDREAM is reported in Appendix C, while detailed training procedures, schedule ablations, training-data accounting, and computational costs are provided in Appendix D. Complete hyperparameter configurations are provided in Appendix E and Appendix F. Detailed per-task Meta-World results are reported in Appendix G. Together with the supplementary code, these materials provide the information needed to reproduce the proposed methods and reported results.

## REFERENCES

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

Sina Alemohammad, Josue Casco-Rodriguez, Lorenzo Luzi, Ahmed Imtiaz Humayun, Hossein Babaei, Daniel LeJeune, Ali Siahkoohi, and Richard Baraniuk. Self-consuming generative models go mad. In *International Conference on Learning Representations*, volume 2024, pp. 53581–53608, 2024.

Sina Alemohammad, Zhangyang Wang, and Richard G Baraniuk. Neon: Negative extrapolation from self-training improves image generation. *arXiv preprint arXiv:2510.03597*, 2025.

Quentin Bertrand, Joey Bose, Alexandre Duplessis, Marco Jiralerspong, and Gauthier Gidel. On the stability of iterative retraining of generative models on their own data. In *International Conference on Learning Representations*, volume 2024, pp. 12137–12165, 2024.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Elvis Dohmatob, Yunzhen Feng, Pu Yang, Francois Charton, and Julia Kempe. A tale of tails: Model collapse as a change of scaling laws. *arXiv preprint arXiv:2402.07043*, 2024.

Benjamin Eysenbach, Alexander Khazatsky, Sergey Levine, and Russ R Salakhutdinov. Mismatched no more: Joint model-policy optimization for model-based rl. *Advances in Neural Information Processing Systems*, 35:23230–23243, 2022.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

Matthias Gerstgrasser, Rylan Schaeffer, Apratim Dey, Rafael Rafailov, Henry Sleight, John Hughes, Tomasz Korbak, Rajashree Agrawal, Dhruv Pai, Andrey Gromov, et al. Is model collapse inevitable? breaking the curse of recursion by accumulating real and synthetic data. *arXiv preprint arXiv:2404.01413*, 2024.

Nate Gillman, Michael Freeman, Daksh Aggarwal, Chia-Hong Hsu, Calvin Luo, Yonglong Tian, and Chen Sun. Self-correcting self-consuming loops for generative model training. *arXiv preprint arXiv:2402.07087*, 2024.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

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. PMLR, 2019b.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Danijar Hafner, Wilson Yan, and Timothy Lillicrap. Training agents inside of scalable world models. *arXiv preprint arXiv:2509.24527*, 2025.

Nick Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, volume 2024, pp. 47376–47405, 2024.

Haoran He, Chenjia Bai, Kang Xu, Zhuoran Yang, Weinan Zhang, Dong Wang, Bin Zhao, and Xuelong Li. Diffusion model is an effective planner and data synthesizer for multi-task reinforcement learning. *Advances in neural information processing systems*, 36:64896–64917, 2023.

Audrey Huang, Adam Block, Dylan Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan Ash, and Akshay Krishnamurthy. Self-improvement in language models: The sharpening mechanism. In *International Conference on Learning Representations*, volume 2025, pp. 76687–76739, 2025.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

Haoxin Lin, Yu-Yan Xu, Yihao Sun, Zhilong Zhang, Yi-Chen Li, Chengxing Jia, Junyin Ye, Jiaji Zhang, and Yang Yu. Any-step dynamics model improves future predictions for online and offline reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 70746–70769, 2025.

Yuejiang Liu, Fan Feng, Lingjing Kong, Weifeng Lu, Jinzhou Tang, Kun Zhang, Kevin Murphy, Chelsea Finn, and Yilun Du. World action verifier: Self-improving world models via forward-inverse asymmetry. *arXiv preprint arXiv:2604.01985*, 2026.

Cong Lu, Philip Ball, Yee Whye Teh, and Jack Parker-Holder. Synthetic experience replay. *Advances in Neural Information Processing Systems*, 36:46323–46344, 2023.

Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample-efficient world models. *arXiv preprint arXiv:2209.00588*, 2022.

Vincent Micheli, Eloi Alonso, and François Fleuret. Efficient world models with context-aware tokenization. *arXiv preprint arXiv:2406.19320*, 2024.

Thanh Nguyen-Tang and Raman Arora. On sample-efficient offline reinforcement learning: Data diversity, posterior sampling and beyond. *Advances in neural information processing systems*, 36: 61115–61157, 2023.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. *Advances in neural information processing systems*, 30, 2017.

Yifu Qiu, Zheng Zhao, Waylon Li, Yftah Ziser, Anna Korhonen, Shay B Cohen, and Edoardo M Ponti. Self-improving world modelling with latent actions. *arXiv preprint arXiv:2602.06130*, 2026.

Marc Rigter, Bruno Lacerda, and Nick Hawes. One risk to rule them all: A risk-sensitive perspective on model-based offline reinforcement learning. *Advances in neural information processing systems*, 36:77520–77545, 2023.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Nicolas Papernot, Ross Anderson, and Yarin Gal. Ai models collapse when trained on recursively generated data. *Nature*, 631(8022):755–759, 2024.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 32145–32168. PMLR, 2023.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

Ren Wang, Kevin Frans, Pieter Abbeel, Sergey Levine, and Alexei Efros. Prioritized generative replay. In *International Conference on Learning Representations*, volume 2025, pp. 47151–47169, 2025.

Guowei Xu, Ruijie Zheng, Yongyuan Liang, Xiyao Wang, Zhecheng Yuan, Tianying Ji, Yu Luo, Xiaoyu Liu, Jiaxin Yuan, Pu Hua, et al. Drm: Mastering visual reinforcement learning through dormant ratio minimization. In *International Conference on Learning Representations*, volume 2024, pp. 56219–56243, 2024.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *Advances in neural information processing systems*, 34:25476–25488, 2021.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pp. 1094–1100. PMLR, 2020a.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in neural information processing systems*, 33:14129–14142, 2020b.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Weipu Zhang, Gang Wang, Jian Sun, Yetian Yuan, and Gao Huang. Storm: Efficient stochastic transformer based world models for reinforcement learning. *Advances in Neural Information Processing Systems*, 36:27147–27166, 2023.

# A SELF-IMPROVING SYNTHETIC EXPERIENCE REPLAY TRAINING DETAILS

In this section, we first visualize the negative self-training direction underlying S$^2$ER and compare it with the direction induced by additional real-data fine-tuning, showing that self-training-induced displacement provides a useful direction for model improvement. We then study the update frequency and held-out real-loss threshold $\gamma_{\text{gate}}$ in S$^2$ER, examining their effects on performance, stability, and computational efficiency.

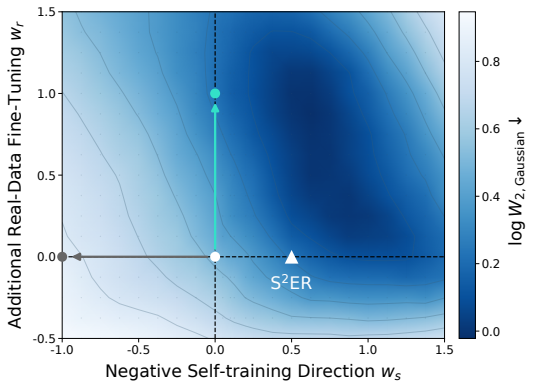



Figure 7: Comparison of the negative self-training direction and the real-data fine-tuning direction.

**Reversed self-training direction versus real-data fine-tuning.** We visualize the negative self-training direction underlying S$^2$ER in Figure 7 using a pretrained Walker2d diffusion checkpoint. We first adapt a reference model $\phi_0$ on 1k real transitions, then fine-tune two copies on self-generated data and on the original 1k plus 4k additional real transitions, obtaining $\phi_s$ and $\phi_r$, respectively. The heatmap shows the log Gaussian 2-Wasserstein distance between generated and held-out real transitions over the parameter plane:

$$\phi(w_s, w_r) = \phi_0 + w_s(\phi_0 - \phi_s) + w_r(\phi_r - \phi_0).$$

Starting from the original reference model $\phi_0$, negative extrapolation to $(w_s, w_r) = (0.5, 0)$ reduces the distance from 1.645 to 1.266, close to the value of 1.248 obtained by fine-tuning with 4k additional real transitions at $(0, 1)$. This diagnostic demonstrates that the self-training displacement identifies a useful correction for the original model, recovering much of the distributional improvement from additional real-data fine-tuning without using those extra transitions.

**Update interval.** Within each S$^2$ER diffusion-model refresh, negative extrapolation is applied once after the diffusion-model update. We first compare refresh intervals of 5k, 10k, and 20k environment steps. Since the 20k schedule underperforms on the Cheetah-Run task, we subsequently focus the ablation on the more competitive 5k and 10k schedules. At the same time, we retain a control that applies S$^2$ER only once at 20k steps to examine whether the gains depend on continued self-improving updates rather than a single extrapolation event. As shown in Figure 8, the 5k schedule achieves the strongest overall performance, while repeated corrections consistently outperform the one-time control. We therefore adopt a 5k refresh interval in the main experiments.

**Gate threshold.** We next fix the update interval to 5k environment steps and vary the held-out real-loss threshold $\gamma_{\text{gate}} \in \{0.05, 0.10, 0.20\}$. Figure 9 shows that performance is relatively insensitive to $\gamma_{\text{gate}}$ within this range, suggesting that S$^2$ER does not depend on a narrowly tuned gate. Among the tested values, $\gamma_{\text{gate}} = 0.10$ provides the strongest and most stable cross-task performance, and is therefore adopted in the main experiments.

The update interval and gate threshold jointly control the effective strength of self-improvement. More frequent updates provide more opportunities to correct the model distribution but increase computational cost, while a larger $\gamma_{\text{gate}}$ permits more aggressive negative extrapolation at the risk of a larger increase in held-out real loss. Our final configuration therefore uses a 5k update interval and $\gamma_{\text{gate}} = 0.10$ to balance performance, stability, and efficiency.

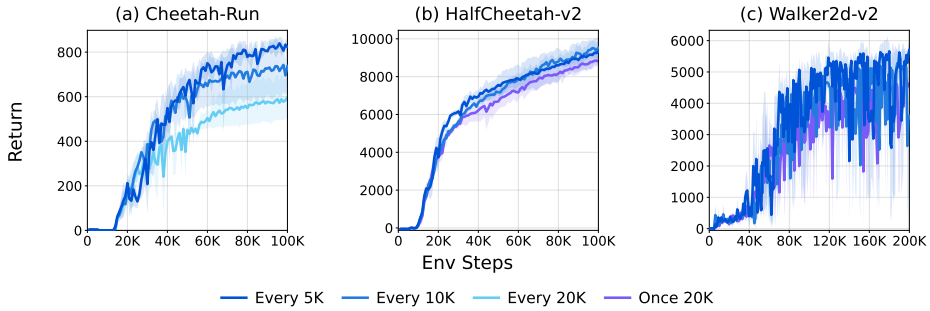



Figure 8: Ablation of the S$^2$ER update interval across three continuous-control tasks over 3 seeds. Shading denotes $\pm 1$ standard deviation.

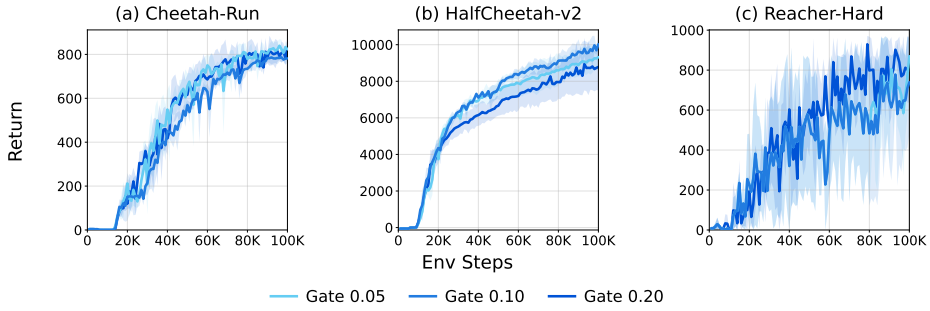



Figure 9: Ablation of the threshold $\gamma_{\text{gate}}$ with a fixed 5k update interval over 3 seeds. Shading denotes $\pm 1$ standard deviation.

**Computational overhead.** We further compare the computational cost of S$^2$ER with SYNTHER and PGR on HalfCheetah-v2. For SYNTHER and PGR, all runtime and memory measurements are obtained using their official implementations under the same hardware and evaluation setting, while S$^2$ER is implemented on top of the same SYNTHER pipeline. As shown in Table 2, on HalfCheetah-v2, S$^2$ER introduces no additional model parameters and only a modest increase in generation VRAM. The self-improving procedure increases total training time by only $4.5\%$ over SYNTHER, while remaining substantially more efficient than PGR in both memory and runtime. These results show that the performance gains of S$^2$ER require only limited computational overhead.

Table 2: Full-run computational cost on HalfCheetah-v2. SYNTHER and PGR measurements are obtained using official implementations under the same hardware and evaluation setting.

| **Metric** | **SYNTHER** | **PGR** | **S$^2$ER (ours)** |
|---|---|---|---|
| Model size ($\times 10^6$ params.) | 6.685 | 26.247 (+292.6%) | 6.685 (+0.0%) |
| Generation VRAM (GB) | 0.537 | 1.002 (+86.7%) | 0.713 (+32.8%) |
| Diffusion training time (hours) | 3.500 | 4.898 (+39.9%) | 3.811 (+8.9%) |
| Generation time (hours) | 0.319 | 1.609 (+404.4%) | 0.327 (+2.5%) |
| Total training time (hours) | 7.020 | 9.707 (+38.3%) | 7.339 (+4.5%) |

**GPU hours.** Across five seeds and six tasks—Walker2d at 200k steps and HalfCheetah, Hopper, Cheetah-Run, Reacher-Hard, and Finger-Turn-Hard at 100k steps—SYNTHER requires 250.23 RTX 3090 GPU-hours, while S$^2$ER requires 254.42 GPU-hours, corresponding to only a 1.7% increase in total computation.

## B A COMPLETE TRAINING INSTANCE OF S$^2$ER

We provide a complete S$^2$ER training instance on Walker2d-v2 to illustrate how the selected model and distributional metrics evolve throughout online training. The diffusion model is refreshed every 5k environment steps. At each refresh, candidate models are first constrained to at most a $10\%$ increase in held-out real denoising loss, and tFID then selects the final candidate over self-training checkpoints $k^\star \in \{0, 100, 250, 500, 1000\}$ and extrapolation weights $w^\star \in \{0, 0.05, 0.1, 0.2, 0.5, 1.0\}$, where candidates with $k^\star = 0$ or $w^\star = 0$ reduce to the unmodified reference model. We save the selected and reference checkpoints throughout training and recompute tFID, precision, and recall using newly sampled real and synthetic transitions for analysis.

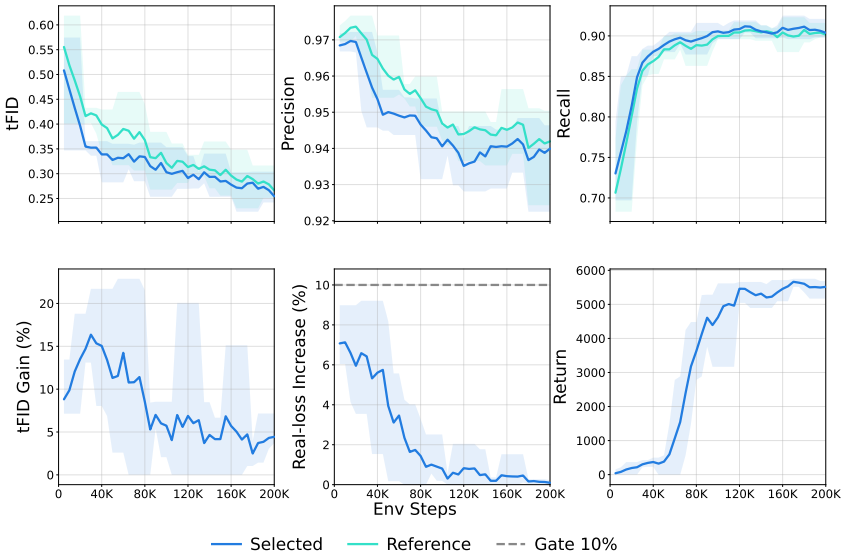



Figure 10: Complete 5k-refresh trajectory of one S$^2$ER training instance on Walker2d-v2.

Figure 10 presents the complete training trajectory. The trajectory reveals a gradual evolution of the synthetic-data distribution as policy learning progresses. Both the reference and selected tFID generally decrease over training, while recall increases and precision exhibits a mild downward trend. This indicates that the synthetic transition distribution becomes increasingly aligned with the evolving real-data distribution, while maintaining broad coverage with only limited changes in distributional precision.

The behavior of S$^2$ER also evolves throughout training. The selected model typically achieves lower tFID than the reference model, although the relative improvement tends to decrease over training. Meanwhile, the selected real-loss increase tends to decline from several percentage points early in training to nearly zero at later stages, indicating that later distributional improvements are achieved with smaller increases in held-out real loss.

This behavior follows from the constrained candidate selection procedure: among candidate corrections that satisfy the prescribed real-loss gate, S$^2$ER selects the one with the lowest tFID on the validation set. Overall, this example illustrates that S$^2$ER can repeatedly improve real–synthetic distributional alignment while controlling changes in held-out real loss.

## C Ablation on Reconstruction Loss in Latent Self-Training

In this section, we examine whether incorporating observation reconstruction into latent self-training provides a more effective negative-extrapolation direction and compare this reconstruction variant with our reconstruction-free objective.

**Motivation.** Our main latent self-training objective omits DreamerV3's observation reconstruction term because imagined rollouts do not provide corresponding ground-truth observations. We therefore examine whether reconstruction from model-generated pseudo-observations can provide an additional useful training signal. Starting from a posterior latent state sampled from the replay buffer, the current RSSM generates an imagined trajectory $\widetilde{\boldsymbol{s}}_{1:H}$. For each imagined latent state $\widetilde{\boldsymbol{s}}_{t+1}$, we decode a pseudo-observation

$$\widetilde{\boldsymbol{o}}_{t+1} = \mathrm{sg}[\mathrm{Dec}_{\phi_{\mathrm{dec}}}(\widetilde{\boldsymbol{s}}_{t+1})]\,, \tag{7}$$

where sg denotes stop-gradient. Thus, $\widetilde{\boldsymbol{o}}_{t+1}$ is treated as a fixed reconstruction target and does not propagate gradients through the rollout that generated it.

Our standard latent self-training objective is

$$\mathcal{L}_{\mathrm{s}} = \mathcal{L}_{\mathrm{dyn}} + \mathcal{L}_{\mathrm{rew}} + \mathcal{L}_{\mathrm{con}}, \tag{8}$$

where the three terms correspond to latent dynamics, reward, and continuation prediction. To introduce reconstruction, the detached pseudo-observation is first re-encoded,

$$\widehat{\boldsymbol{e}}_{t+1} = \mathrm{Enc}_{\phi_{\mathrm{enc}}}(\widetilde{\boldsymbol{o}}_{t+1}), \qquad \widehat{\boldsymbol{z}}_{t+1} \sim q_{\phi_{\mathrm{rssm}}}(\boldsymbol{z}_{t+1} \mid \widetilde{\boldsymbol{h}}_{t+1}, \widehat{\boldsymbol{e}}_{t+1}), \tag{9}$$

forming the reconstructed latent state $\widehat{\boldsymbol{s}}_{t+1} = (\widetilde{\boldsymbol{h}}_{t+1}, \widehat{\boldsymbol{z}}_{t+1})$. The frozen decoder then evaluates

$$\mathcal{L}_{\mathrm{rec}} = -\mathbb{E}\left[\log p_{\phi_{\mathrm{dec}}}(\widetilde{\boldsymbol{o}}_{t+1} \mid \widehat{\boldsymbol{s}}_{t+1})\right]. \tag{10}$$

The reconstruction variant therefore optimizes

$$\hat{\mathcal{L}}_{\mathrm{s}} = \mathcal{L}_{\mathrm{s}} + \lambda_{\mathrm{rec}}\mathcal{L}_{\mathrm{rec}}, \tag{11}$$

with $\lambda_{\mathrm{rec}} = 1$. In both variants, the encoder, decoder, reward predictor, continuation predictor, actor, and critic remain frozen; gradients from all objectives propagate only to the RSSM parameters $\phi_{\mathrm{rssm}}$, which subsequently receive the same negative-extrapolation correction.

**Experiment.** We evaluate both self-training objectives on the image-based Meta-World Box-Close task for 500k environment steps. Both variants use the same DreamerV3 training configuration and self-improving schedule, with self-training imagination horizon $H = 5$, extrapolation weight $w = 0.5$, and one self-improving update every 10 standard Dreamer updates. All other settings are kept identical between the two variants.

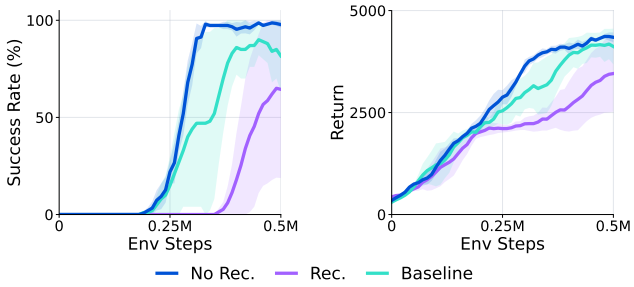



Figure 11: Effect of the reconstruction objective on latent self-training in Meta-World Box-Close.

**Results.** Figure 11 shows that omitting the reconstruction objective leads to faster learning and better data efficiency. The standard latent self-training objective consistently outperforms both the reconstruction variant and the baseline, while the reconstruction variant exhibits a clear delay in learning and consistently lower final performance across all three seeds.

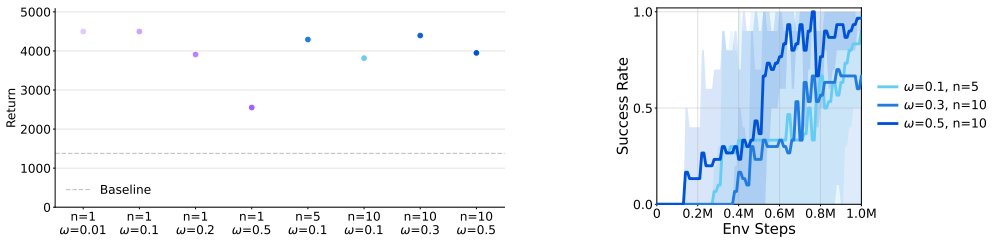
These results support omitting the reconstruction term from latent self-training. When imagined rollouts remain entirely in the latent space without decoding, the reconstruction branch is absent from the computation graph and therefore contributes zero gradient to the RSSM. Empirically, the objective in Eq. (6) provides a cleaner and more informative direction for negative extrapolation: the reconstruction-free variant learns substantially faster and achieves higher data efficiency, whereas adding the reconstruction loss results in notably lower performance than both the reconstruction-free variant and the Dreamer baseline.

This ablation further suggests that the self-training objective need not reproduce every component of the full Dreamer world-model objective to yield a useful extrapolation direction. Even without reconstruction, reversing the displacement induced by the remaining synthetic objectives is sufficient to improve control performance in this ablation.

## D REDREAM TRAINING DETAILS

**Ablation on negative extrapolation weight and interval.** We investigate how the extrapolation weight $w$ and correction interval $n$ affect REDREAM, with the goal of balancing training effectiveness and computational efficiency. Our initial configuration applies a mild correction with $w = 0.1$ after every Dreamer update ($n = 1$). As shown in Figure 13, this frequent correction achieves strong task-median performance and can accelerate learning, but incurs substantially higher computational overhead, as quantified in Table 4.

We therefore perform a targeted schedule sweep on the Meta-World Sweep task. Figure 12 shows that all tested negative-extrapolation schedules outperform the Dreamer baseline, indicating that the gains are not confined to a narrow hyperparameter choice. Under $n = 1$, mild extrapolation performs well, whereas increasing the weight to $w = 0.5$ degrades performance, suggesting that overly strong corrections become detrimental when applied too frequently. Motivated by both this behavior and the computational cost of $n = 1$, we reduce the correction frequency and compare $w \in \{0.1, 0.3, 0.5\}$ over multiple seeds on Sweep. Among these settings, $n = 10, w = 0.5$ provides the strongest overall performance on Sweep while reducing the number of corrections by an order of magnitude relative to $n = 1$, and we therefore use it as the default configuration.




(a) Comparison of extrapolation weights $w$ and intervals $n$.

(b) SR curves of the selected schedules.

Figure 12: Ablation of the negative-extrapolation weight $w$ and interval $n$ on Meta-World Sweep.

After fixing this default configuration on Sweep, we further compare $n = 1, w = 0.1$ and $n = 10, w = 0.5$ across 10 additional Meta-World tasks as a broader robustness analysis. Although the frequent mild schedule attains a higher task-median success rate late in training, the $n = 10, w = 0.5$ configuration maintains competitive task-mean performance and exhibits more consistent per-task improvements in Figure 14. This broader evaluation confirms that the schedule selected on Sweep remains effective across diverse tasks while retaining a good trade-off between performance and computational cost.

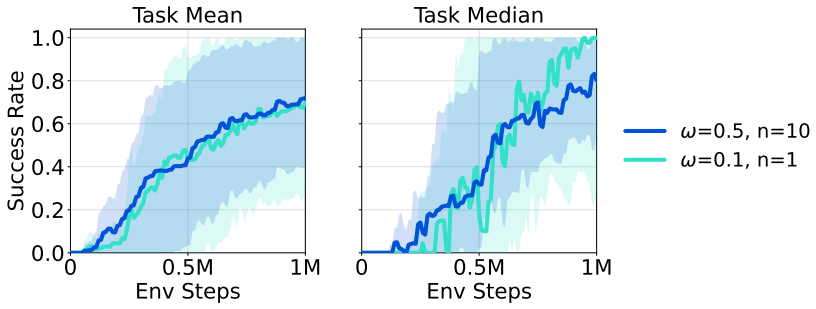



Figure 13: Effect of negative-extrapolation strength and interval on the mean and median success rates across 10 additional Meta-World tasks.

**Training-data accounting.** Negative extrapolation is integrated into online Dreamer training while environment interaction, replay collection, and standard Dreamer updates proceed unchanged. We run 16 parallel environments with an action repeat of two. Each Dreamer update samples 16 replay sequences of length 65, where the first position initializes the recurrent state and the remaining 64 positions are used for optimization, yielding 1,024 real replay positions per update.

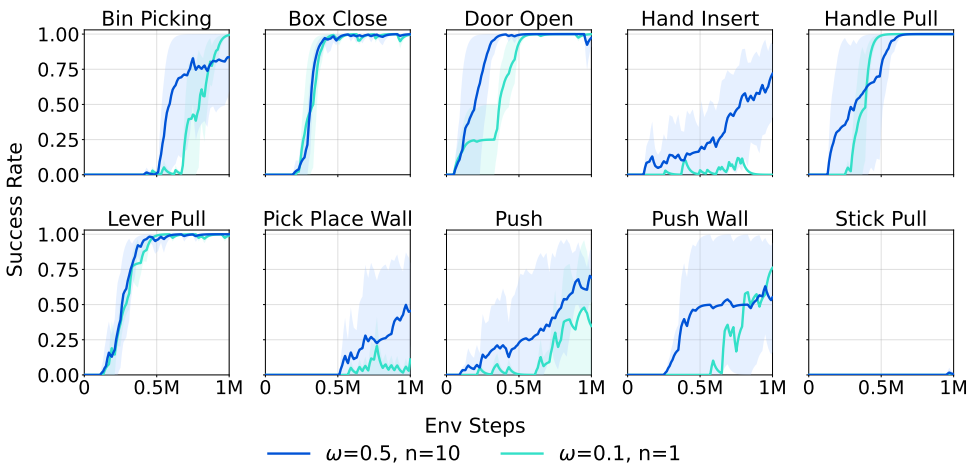



Figure 14: Comparison of two negative-extrapolation schedules across 10 additional tasks.

When a negative-extrapolation correction is scheduled, the posterior states from the same Dreamer batch are detached and used as rollout starting points. We retain at most 1,024 starting states and generate five imagined transitions from each, yielding 5,120 synthetic transitions per correction. Their next-state, reward, and continuation predictions are detached and treated as fixed targets for the synthetic objectives. These transitions are used only for the current correction and are discarded immediately afterward without being added to replay. When no correction is scheduled, no additional self-training rollouts or self-training updates are performed. Table 3 summarizes the cumulative transition counts for world-model training at 500k environment steps.

Table 3: Processed training transitions at 500k environment steps.

| **Method** | **Real replay** | **Synthetic** | **Total** |
|---|---|---|---|
| **Dreamer** | 127.452M | 0 | 127.452M |
| **Every update,** $n = 1$ | 127.452M | 637.261M | 764.713M |
| **Every 10 updates,** $n = 10$ | 127.452M | 63.724M | 191.176M |

**Computational cost.** Table 4 reports a controlled single-task wall-clock comparison. The computational overhead decreases substantially when corrections are applied less frequently. Applying corrections after every update ($n = 1$) increases training time by 28.99%, while the $n = 10$ setting used in our main experiments incurs only a 1.93% overhead.

Table 4: Measured runtime of REDREAM on an NVIDIA A100 GPU for Sweep at 1.01M environment steps. Overhead is measured relative to DreamerV3 (7.481 h).

| **Correction interval** | **Runtime (h)** | **Overhead** |
|---|---|---|
| **Every update,** $n = 1$ | 9.650 | +28.99% |
| **Every 10 updates,** $n = 10$ | 7.625 | +1.93% |

**GPU hours.** The full three-seed experiments require approximately 268.5 A100 GPU-hours across the 12 Meta-World tasks, with an overall computational overhead of 5.6% over the original DreamerV3 baseline. These costs include periodic evaluation during training but exclude hyperparameter searches, ablation studies, and failed runs.

## E $S^2$ER HYPERPARAMETERS

In this section, we provide the complete $S^2$ER hyperparameters in Table 5.

Table 5: Hyperparameters for $S^2$ER on six continuous-control tasks from MuJoCo and DMC.

| Component | Hyperparameter | Value |
|---|---|---|
| *Benchmarks* | | |
| | **Gym MuJoCo tasks** | Walker2d-v2, HalfCheetah-v2, Hopper-v2 |
| | **DMC tasks** | Cheetah-Run, Reacher-Hard, Finger-Turn-Hard |
| | **Environment steps** | 200k (Walker2d); 100k (others) |
| | **Model terminal transitions** | Yes (MuJoCo); No (DMC) |
| | **Reward normalization** | Yes (MuJoCo); No (DMC) |
| SYNTHER | | |
| | **Real-data ratio** $r$ | 0.5 (128 real and 128 synthetic) |
| | **Real replay capacity** $\lvert\mathcal{D}_{\mathrm{real}}\rvert$ | 1M |
| | **Synthetic replay capacity** $\lvert\mathcal{D}_{\mathrm{syn}}\rvert$ | 1M |
| | **Synthetic transitions per refresh** | 200k |
| | **Diffusion training updates** | 100k per refresh |
| | **Diffusion batch size** | 256 |
| | **Diffusion learning rate** | $3 \times 10^{-4}$ |
| | **Diffusion architecture** | 6-layer ResMLP, width 1024 |
| | **Diffusion sampling steps** | 128 |
| | **Data scale** $\sigma_{\mathrm{data}}$ | 1.0 |
| | **Policy update-to-data ratio** | 20 |
| | **Initial random transitions** | 5k |
| $S^2$ER *(ours)* | | |
| | **Refresh interval** $n$ | 5k environment steps |
| | **Self-training set size** | 50k |
| | **Self-training updates** | 1k |
| | **Self-training checkpoints** $\mathcal{K}$ | $\{0, 100, 250, 500, 1000\}$ |
| | **Extrapolation weights** $\mathcal{W}$ | $\{0, 0.05, 0.1, 0.2, 0.5, 1.0\}$ |
| | **Fidelity tolerance** $\gamma_{\mathrm{gate}}$ | 0.10 |
| | **Held-out real transitions** | 1k |
| | **Real-loss evaluations** | 4 repetitions |
| | **Distributional-metric samples** | 1k real and 10k synthetic |
| *Checkpointing* | | |
| | **Diffusion checkpoints** | Every 5k environment steps |
| | **Policy checkpoints** | Every 5k environment steps |
| | **Candidate checkpoints** | All $(k, w) \in \mathcal{K} \times \mathcal{W}$ |

# F REDREAM HYPERPARAMETERS

In this section, we provide the complete REDREAM hyperparameters in Table 6.

Table 6: Hyperparameters for the Dreamer backbone and REDREAM.

| Component | Hyperparameter | Value |
|---|---|---|
| *Dreamer backbone* | | |
| | **Observation resolution** | $64 \times 64$ |
| | **Parallel environments** | 16 |
| | **Replay-buffer capacity** | $5 \times 10^5$ transitions |
| | **Training batch** | 16 sequences of length 65 |
| | **Training ratio** | 512 gradient samples per agent step |
| | **Action repeat** | 2 |
| | **Evaluation interval** | $10^4$ environment steps |
| | **Evaluation episodes** | 10 |
| | **Imagination horizon** | 15 |
| | **Return parameter $\lambda$** | 0.95 |
| | **Actor entropy coefficient** | $3 \times 10^{-4}$ |
| | **Free-nats threshold** | 1.0 |
| | **Optimizer** | LaProp |
| | **Learning rate** | $4 \times 10^{-5}$ |
| | **Optimizer coefficients** | $(\beta_1, \beta_2) = (0.9, 0.999)$ |
| | **Learning-rate warm-up** | $10^3$ updates |
| | **Adaptive gradient clipping** | 0.3 |
| | **Slow-target update rate** | 0.02 |
| *World model* | | |
| | **Deterministic state size** | 2048 |
| | **Stochastic state** | 32 categorical variables with 16 classes |
| | **Hidden size** | 256 |
| | **RSSM blocks** | 8 |
| | **Activation** | SiLU |
| | **Unimix ratio** | 0.01 |
| REDREAM *(ours)* | | |
| | **Rollout horizon $H$** | 5 |
| | **Self-training interval $n$** | 10 |
| | **Maximum latent starts** | 1024 |
| | **Self-training size $\eta$** | $1024 \times 5 = 5120$ |
| | **Self-training learning rate $\alpha_{\mathrm{st}}$** | $4 \times 10^{-5}$ |
| | **Extrapolation weight $w$** | 0.5 |
| | **Dynamics-loss coefficient** | 1.0 |
| | **Reward-loss coefficient** | 1.0 |
| | **Continuation-loss coefficient** | 1.0 |

## G Detailed Results on Meta-World

This section presents the learning curves for each of the 12 challenging Meta-World tasks used in our evaluation. Detailed per-task results are shown in Figure 15.

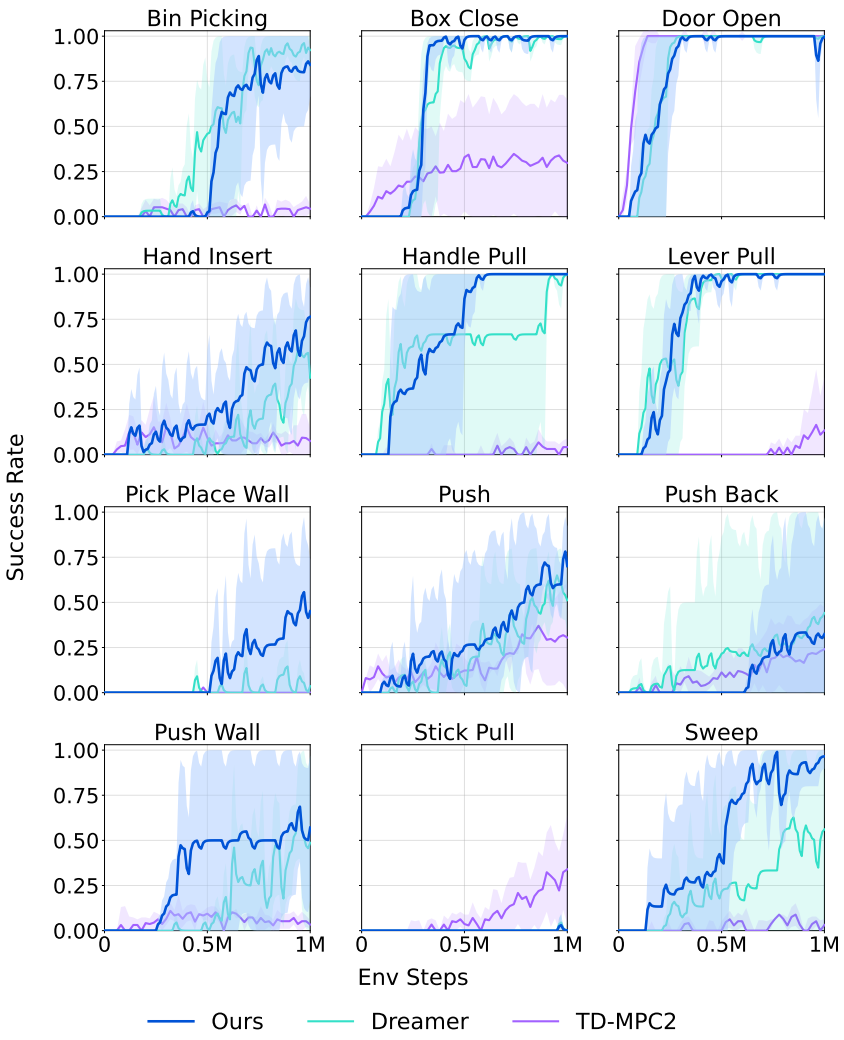



Figure 15: Detailed learning curves for the 12 Meta-World tasks.