Stealthy World Model Manipulation via Data Poisoning

Yibin Hu
Department of Computer Science
Tulane University
New Orleans, LA, USA
yhu20@tulane.edu

Zizhan Zheng
Department of Computer Science
Tulane University
New Orleans, LA, USA
zzheng3@tulane.edu

Abstract

Model-based learning agents use learned world models to predict future states, plan actions, and adapt to new environments. However, updating world models from collected experience creates a training-time attack surface: poisoned fine-tuning trajectories can manipulate learned dynamics and corrupt downstream planning. We propose SWAAP, a two-stage data poisoning framework for learned world models. The first stage identifies a harmful target world model that induces low-return behavior while remaining close to clean dynamics, using a first-order bilevel approximation motivated by a transition-gradient theorem. The second stage constructs poisoned transition targets through gradient matching, aligning the victim's fine-tuning gradients with the target-model update. A prediction-error regularizer controls the data-level deviation by matching the clean model's natural prediction residuals. Experiments across continuous-control benchmarks demonstrate substantial return reductions while maintaining data deviations near the clean reference scale at stealth-prioritized settings. Residual screening and CUSUM detect several attacks at lower stealth weights, while stronger regularization suppresses their detection signal. Robust fine-tuning with TRIM recovers performance in some settings, but leaves substantial attack effects or incurs clean-utility losses in others. Additional evaluations on DINO-WM and a multi-task victim demonstrate poisoning across world-model architectures and encoders. These results establish world-model fine-tuning as a security-relevant part of the planning pipeline and motivate defenses that protect both adaptation data and learned dynamics. Code is available at https://github.com/yibhu/SWAAP.

1 Introduction

While artificial intelligence (AI) has achieved remarkable success across various domains, building general-purpose agents that can quickly adapt to new tasks remains a major challenge, particularly for sequential decision-making tasks in open-ended environments that require substantial planning and adaptation. A promising direction is the development of *world models* [Ha and Schmidhuber, 2018] that accurately capture environmental structure and dynamics to support a wide range of downstream tasks. In this context, predictive world models, which allow agents to "imagine" future scenarios for safer, more efficient decisions, are proliferating [Hafner et al., 2024, Hansen et al., 2024, Zhou et al., 2025]. Further, with recent advances in diffusion and transformer architectures, foundation world models [OpenAI, 2025, Bruce et al., 2024, Nvidia, 2025a] capable of simulating interactive environments from multi-modal input are emerging and are increasingly being applied in complex domains such as autonomous vehicles and robotics [Nvidia, 2025b], making them valuable targets for malicious attacks.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

To support effective decision-making across diverse domains, world models must encode broad knowledge, process high-dimensional inputs (e.g., images, videos, and text), and make long-horizon predictions, introducing new vulnerabilities not present in traditional supervised learning or model-free reinforcement learning (RL) systems. Despite extensive research in AI security and adversarial machine learning over the past decade, ensuring the robustness of world models against adversarial manipulation remains largely unexplored, limiting their deployment in high-stakes domains.

In this work, we take an early step toward adversarially robust world modeling by introducing poisoning attacks tailored to world models. Our method strategically alters trajectory data used for training or fine-tuning, with the objective of manipulating model-based decision-making while maintaining outputs close to those of a clean model to evade detection. This setting exposes a training-time failure mode specific to world-model adaptation, which is a fundamental vulnerability in world models that underpins their reliability in downstream applications.

Traditional data poisoning methods from supervised learning [Biggio et al., 2012, Muñoz-González et al., 2017, Geiping et al., 2021] cannot be directly applied to our setting. These approaches typically assume fully differentiable training pipelines and discrete labels (e.g., flipping a *cat* to a *dog*), allowing the adversary to optimize per-example perturbations via gradient-based techniques. In contrast, poisoning a world model requires identifying perturbations to the predicted next-state outcomes, which are structured, high-dimensional, and may be deterministic or stochastic. Such perturbations influence not only one-step predictions but also compound over long-horizon rollouts. Moreover, differentiating through the training process of the world model is computationally expensive and generally intractable, making standard supervised learning attacks unsuitable for this problem.

Existing data-poisoning attacks in reinforcement learning [Rakhsha et al., 2020, Zhang et al., 2020] primarily manipulate rewards or transitions observed during training to steer the learned policy toward an adversary-specified target policy or behavior. Although performance degradation can in principle be achieved by enforcing a low-return target policy, this perspective does not directly apply to world-model poisoning, where the adversary cannot directly enforce the deployed policy. Instead, a world-model poisoning must act indirectly: it changes fine-tuning transitions so that the learned dynamics induce harmful planning or model-based policy improvement. SWAAP operationalizes this indirect attack by identifying a poisoned world model under which harmful behavior becomes favorable for the victim's planner, and then realizing this model by poisoning only a bounded fraction of fine-tuning transitions. This creates a coupled inverse-planning and data-realization problem: even if a low-return target policy is known, there may be no nearby world model that induces it while remaining sufficiently close to clean dynamics to evade detection. Moreover, existing RL poisoning formulations are designed for simpler tabular or low-dimensional settings and do not directly scale to deep world models with high-dimensional continuous latent states, continuous actions, and long-horizon compounding prediction errors, as in TD-MPC2 [Hansen et al., 2024], DINO-WM [Zhou et al., 2025], and DreamerV3 [Hafner et al., 2024]. These differences motivate a scalable poisoning framework tailored to learned world models.

In this work, we propose **SWAAP** (**S**tealthy **W**orld Model M**A**nipulation via D**A**ta **P**oisoning), a two-stage data poisoning framework to manipulate learned world models in model-based RL. Our formulation covers stochastic and deterministic transition dynamics. SWAAP first identifies a stealthy target world model that induces low-return behavior while staying close to the intact world model, via bilevel optimization. It then optimizes a small fraction of the fine-tuning transition targets to induce harmful changes in the victim world model. Our contributions can be summarized as follows.

- We formulate data poisoning of learned world-model dynamics in deep model-based RL, targeting downstream planning while limiting deviation from clean-model predictions.
- **Two-stage decomposition**. Direct bilevel optimization through fine-tuning is computationally demanding. We decouple it into (i) finding a stealthy adversarial target world model enabled by a transition gradient, and (ii) realizing it via gradient-matched data poisoning applied to trajectory data.
- **Stealth-constrained data realization.** We adapt gradient matching to world-model fine-tuning by constructing poisoned transition targets whose gradients steer the victim toward the Stage 1 target while a prediction-error regularizer encourages residuals close to the clean model's natural deviation scale. We also introduce a trajectory-consistent variant for sequential fine-tuning data, where a poisoned intermediate state is used consistently as both the next-state target at time t and the current state at time $t+1$.

- **Empirical evaluation.** We evaluate SWAAP across DMControl [Tassa et al., 2018], MyoSuite [Caggiano et al., 2022], MetaWorld [Yu et al., 2020] and ManiSkill2 [Gu et al., 2023], comparing poisoning strategies and testing residual screening, CUSUM and TRIM. The results demonstrate harmful fine-tuning at data deviations near the clean reference scale and characterize the protection provided by the evaluated defenses. DINO-WM and gray-box experiments examine transfer across world-model architectures and encoders.

2 System and Threat Models

In this section, we present the system and threat models, covering both world models and our proposed data-poisoning attacks framework. A detailed summary of related work on world models, model-based RL, model poisoning, data poisoning, and defenses is provided in Appendix C.

2.1 World Models

We study an agent interacting with an environment formalized as a Markov decision process (MDP) $(S, A, P, R, \gamma, \mu)$, where μ is the initial state distribution, S is the state space, A is the action space, $P : S \times A \to \Delta(S)$ is the transition kernel, $R : S \times A \times S \to \mathbb{R}$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor. The agent learns a world model P_ψ (parameterized by ψ) that approximates the environment dynamics $P(s' \mid s, a)$ from transition data $D = \{(s_i, a_i, s'_i)\}_{i=1}^{|D|}$. To simplify notation and keep the discussion general, we assume a deterministic transition function in the main text and write the next state as $s' = P_\psi(s, a)$. With this slight abuse of notation, P_ψ is trained by minimizing the squared prediction error

$$\mathcal{L}(\psi; D) = \sum_{(s,a,s') \in D} \|s' - P_\psi(s, a)\|_2^2.$$

Extensions to the stochastic transition functions are in Appendix D.3, where the squared-error objective is replaced by a likelihood-based formulation. In modern model-based agents, prediction and planning are usually performed in a learned latent space $z = \mathrm{enc}(s)$. To simplify notation, we write transitions in state space, but in our implementation s denotes the encoded latent z unless otherwise stated. Transition predictions, poisoned targets, and stealth metrics are therefore computed in latent space. If fine-tuning data are stored as raw observations, the same objective can in principle be optimized through the encoder by replacing a raw next observation $\tilde{o}'$ whose encoding realizes the desired latent target. Appendix H.2 gives initial raw-space transfer evidence across different encoders.

We adopt TD-MPC2 [Hansen et al., 2024], a representative world modeling framework, to explain our approach. TD-MPC2 jointly trains a latent dynamics model, value functions, and a policy, and uses the policy to initialize model predictive control (MPC). In general, a learned world model induces a downstream policy or planner $\pi_{\theta^\star(\psi)}$ by optimizing expected return under P_ψ:

$$\theta^\star(\psi) \in \arg\max_\theta J(P_\psi, \pi_\theta), \qquad J(P_\psi, \pi_\theta) = \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim P_\psi, \pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t R(s_t, a_t, s_{t+1}) \right].$$

For MPC-based agents, at time t the planner evaluates candidate action sequences $a_{t:t+H-1}$ by rolling them out under P_ψ from $\hat{s}_t = s_t$ and executes only the first action of the best sequence:

$$a^\star_{t:t+H-1} \in \arg\max_{a_{t:t+H-1}} \sum_{h=0}^{H-1} \gamma^h R(\hat{s}_{t+h}, a_{t+h}, \hat{s}_{t+1+h}), \quad \hat{s}_{t+h+1} \sim P_\psi(\cdot \mid \hat{s}_{t+h}, a_{t+h}), \quad a_t^{\mathrm{exec}} = a_t^\star.$$

A detailed description of the MPC procedure is given in Appendix E.1.

2.2 Threat Model

We consider a continual fine-tuning pipeline for learned world models: a developer pretrains a world model on a large general-purpose dataset and then periodically adapts it using locally collected trajectories from a deployed robot, edge device, or simulation client. In such pipelines, compromising the local fine-tuning dataset is a more accessible attack surface than compromising the pre-training process or directly overwriting deployed model parameters. We consider a data-poisoning adversary

with access to this local fine-tuning stream, such as through a compromised data-logging path, data-aggregation service, or insider write access. The attacker may (i) modify at most $r_p|D|$ ($r_p \in (0, 1]$) transitions in the fine-tuning dataset D by replacing the recorded next state s'_i with an adversarial target $\tilde{s}'_i$, leaving s_i, a_i, and the reward unchanged; and (ii) interact with a clean copy of the environment to collect reference trajectories for surrogate fitting and stealth regularization. The attack objective is to reduce the long-term return of the agent after the poisoned world model is used for planning or model-based policy improvement.

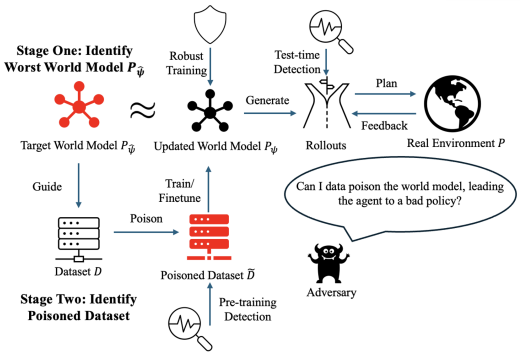


Figure 1: SWAAP pipeline.

In our main experiments, we assume white-box access to the victim world-model architecture, pretrained weights ψ_0, and fine-tuning loss. The attacker may additionally collect clean reference trajectories from the environment for surrogate fitting and stealth regularization. We use this as an attack feasibility study and a worst-case robustness evaluation, analogous to white-box adversarial evaluation in supervised learning; it is also relevant to open-weight world models and downstream adaptation settings. We also evaluate a gray-box variant in Appendix H.2, in which the attacker does not know the exact victim parameters or the downstream task. In both white-box and gray-box settings, the attacker only modifies a bounded subset of recorded fine-tuning transitions. The attacker does not directly change the deployed model parameters at test time; the attack must instead act through subsequent world-model adaptation. This separates SWAAP from direct model overwrite baselines and from test-time observation perturbations, which intervene at different stages of the pipeline.

2.3 Bilevel Formulation of Problem

The adversary can pursue multiple objectives when manipulating the world model update, such as steering the agent toward a specific target policy or degrading its performance. In this work, we focus on the objective of reducing the agent's long-term return. We consider a data-poisoning adversary that may perturb a bounded fraction of transitions in the fine-tuning dataset D, replacing (s_i, a_i, s'_i) with $(s_i, a_i, \tilde{s}'_i)$ to form a poisoned dataset $\tilde{D}$. After the agent trains on $\tilde{D}$, the resulting world model P_ψ deviates from the true dynamics P and induces suboptimal behavior. Our formulation follows the bilevel structure used in supervised data poisoning, where poisoned examples are outer variables and victim training defines the inner response. MetaPoison is a representative example [Huang et al., 2020]. Our outer objective is closed-loop return. The victim response includes both world-model adaptation and the planner or policy induced by the adapted model. SWAAP approximates these coupled responses through target-model identification followed by gradient-matched data realization. This interaction can be formalized as the following bilevel optimization problem:

$$\begin{aligned} &\min_{\tilde{D}} \quad J\big(P, \pi_{\theta^*(\psi)}\big) + \lambda L(P_\psi, P) \\ &\text{s.t.} \quad P_\psi = F(P_{\psi_0}, \tilde{D}), \theta^*(\psi) = \arg\max_\theta J(P_\psi, \pi_\theta), \sum_{i=1}^{|D|} \mathbf{1}[\tilde{s}'_i \neq s'_i] \leq r_p\,|D|. \end{aligned} \tag{1}$$

where $J\big(P, \pi_{\theta^*(\psi)}\big)$ is the return of the surrogate policy in the true environment, $F(P_{\psi_0}, \tilde{D})$ denotes fine-tuning the initial world model P_{ψ_0} on the poisoned dataset $\tilde{D}$, and $L(P_\psi, P)$ penalizes deviation between the poisoned world model and the true dynamics. In the general stochastic setting, this deviation can be measured by an expected divergence between transition kernels, such as $D_{\mathrm{KL}}(P(\cdot \mid s, a) \,\|\, P_\psi(\cdot \mid s, a))$. Our experiments use deterministic latent world models, where the transition model outputs a point prediction and the KL divergence is degenerate. We therefore instantiate $L(P_\psi, P)$ with the squared latent prediction-error surrogate:

$$L(P_\psi, P) = \mathbb{E}_{(s,a,s')\sim P, \pi_{\theta^*(\psi)}} \left[\|P_\psi(s, a) - s'\|_2^2 \right]. \tag{2}$$

Thus, we instantiate a deterministic formulation here. Appendix D.3 gives the stochastic formulation. In summary, the *outer minimization* reduces real-environment performance, while the *inner maximization* captures the agent's policy optimization under the poisoned world model.

3 Stealthy World Model Manipulation via Data Poisoning

Directly solving Eq. 1 is intractable because it requires optimizing which transition targets to perturb, how to perturb them, and how the victim world model changes after fine-tuning. SWAAP therefore decomposes the attack into two stages, shown in Figure 1. Stage 1 identifies a harmful but low-deviation target world model $\hat{\psi}$ by solving a model-space bilevel problem. Stage 2 realizes this target through gradient-matched data poisoning, constructing $\tilde{D}$ so that fine-tuning gradients steer the victim model toward $\hat{\psi}$. Stealthiness is enforced at both levels: Stage 1 regularizes model-level deviation, while Stage 2 regularizes data-level deviations so poisoned targets remain close to the clean model's natural prediction error.

3.1 Stage 1: Identification of Perturbed Models

In the first stage, the attacker searches for a target world model $\hat{\psi}$ that induces low return when used for planning, while remaining close to the true dynamics. Let $J(P_\psi, \pi_\theta)$ denote the expected return of policy π_θ in the imagined environment P_ψ, and let $J(P, \pi_\theta)$ denote its return in the true environment. The attacker then solves the model-space bilevel problem:

$$\hat{\psi} \in \arg\min_{\psi} \; J(P, \pi_{\theta^\star(\psi)}) + \lambda L(P_\psi, P) \quad \text{s.t.} \quad \theta^\star(\psi) \in \arg\max_{\theta} J(P_\psi, \pi_\theta), \tag{3}$$

where $L(P_\psi, P)$ penalizes deviation from true dynamics and controls the attack–stealth trade-off. Directly solving Equation 3 is difficult because changing ψ affects the agent only indirectly through long-horizon model-based planning or policy improvement. The outer objective is a trajectory-level return evaluated in the true environment, while the inner problem is a nonconvex RL/planning problem over high-dimensional policies and learned dynamics. Thus, applying a generic bilevel optimizer would require differentiating through the implicit response $\theta^\star(\psi)$, which is computationally prohibitive in modern world-model agents. We therefore use a first-order dynamic-barrier bilevel method in the spirit of BOME [Liu et al., 2022], which replaces the implicit optimality condition with a value-function constraint, as detailed in Appendix D.2. However, directly instantiating this update in model-based RL still requires the transition-model gradient $\nabla_\psi J(P_\psi, \pi_\theta)$, which is not provided by the standard policy-gradient theorem. We thus derive a general transition-gradient estimator for stochastic transition kernels $P_\psi(\cdot \mid s, a)$.

Theorem 3.1 (Transition Gradient) *For a finite-horizon MDP with stochastic transition kernel $P_\psi(\cdot \mid s, a)$ and fixed policy π_θ, define*

$$V_t(s) \coloneqq \mathbb{E}_{P_\psi, \pi_\theta}\left[\sum_{i=0}^{T-t-1} \gamma^i R(s_{t+i}, a_{t+i}, s_{t+i+1}) \,\middle|\, s_t = s \right], \qquad 0 \le t < T,$$

with $V_T(s) = 0$. Assuming $P_\psi(s'|s, a)$ is differentiable in ψ, the gradient of the return with respect to the transition parameters is

$$\nabla_\psi J(P_\psi, \pi_\theta) = \mathbb{E}_{(s_t, a_t, s_{t+1}) \sim P_\psi, \pi_\theta}\Big[\sum_{t=0}^{T-1} \gamma^t \big(R(s_t, a_t, s_{t+1}) + \gamma V_{t+1}(s_{t+1}) \big) \cdot \nabla_\psi \log P_\psi(s_{t+1} \mid s_t, a_t) \Big].$$

Using Theorem 3.1, Stage 1 alternates between approximating the agent's response under the current world model and updating (P_ψ, π_θ) with the dynamic-barrier direction. At iteration k, we first run W policy-gradient steps to obtain an approximate best response θ_k^W under P_{ψ_k}. We then estimate the outer and constraint gradients using rollouts from the true environment and the current/perturbed world models, and update $(P_{\psi_k}, \pi_{\theta_k})$ by the first-order dynamic-barrier step. Algorithm 1 summarizes the procedure (see Appendix E.3 for the algorithm details); Appendix D.2 provides the full BOME derivation and Appendix D.1 proves Theorem 3.1, including the discounted infinite-horizon stochastic form and the deterministic finite-horizon specialization used by deterministic latent predictors.

3.2 Stage 2: Poisoning Data to Manipulate Model

Stage 1 identifies a harmful target world model $P_{\hat{\psi}}$. Stage 2 seeks to induce its effect through bounded data poisoning. Let $F(P_{\psi_0}, \tilde{D})$ denote fine-tuning the initial world model on poisoned data $\tilde{D}$. The

attack objective is

$$\min_{\tilde{D}} \; L\big(P_{\hat{\psi}}, F(P_{\psi_0}, \tilde{D})\big) \quad \text{s.t.} \sum_{i=1}^{|D|} \mathbf{1}[\tilde{s}'_i \neq s'_i] \leq r_p \, |D|, \tag{4}$$

where the attacker modifies at most an r_p fraction of next-state targets while keeping (s, a, r) unchanged. Directly solving this problem requires both discrete subset selection and continuous target optimization while repeatedly fine-tuning the victim model, so we use a two-step approximation.

First, let d be the latent dimension, We select the $r_p|\tilde{D}|$ transitions with the largest prediction disagreement between the pretrained and target models,

$$e_i = \frac{1}{d}\big\|P_{\hat{\psi}}(s_i, a_i) - P_{\psi_0}(s_i, a_i)\big\|_2^2.$$

Let D_p be the selected transitions, and $D_c = D \setminus D_p$. The poisoned dataset is $\tilde{D} = D_c \cup \tilde{D}_p$, where only the next-state targets in D_p are editable.

Second, we optimize these targets by matching dynamics-parameter gradients at the pretrained parameters. Define

$$G_{\text{real}} = \nabla_{\psi} \frac{1}{|\tilde{D}|d} \sum_{(s,a,\tilde{s}') \in \tilde{D}} \|P_{\psi}(s, a) - \tilde{s}'\|_2^2 \Big|_{\psi=\psi_0},$$

$$G_{\text{target}} = \nabla_{\psi} \frac{1}{|D_{\text{all}}|d} \sum_{(s,a) \in D_{\text{all}}} \|P_{\psi}(s, a) - P_{\hat{\psi}}(s, a)\|_2^2 \Big|_{\psi=\psi_0}.$$

Here D_{all} is a separate clean reference dataset. The first gradient includes all adaptation rows, and the second describes fitting the pretrained model to the target model's predictions. The model parameters remain fixed while we optimize the poisoned targets.

Following gradient matching [Geiping et al., 2021], we combine cosine alignment with a prediction-error regularizer. Writing $p_i = P_{\psi_0}(s_i, a_i)$, the objective is

$$L_P = (1 - \alpha)\big(1 - \cos(G_{\text{real}}, G_{\text{target}})\big) + \frac{\alpha}{r_p|\tilde{D}|} \sum_{i \in S} \Big(\|\tilde{s}'_i - p_i\|_2 - \|s'_i - p_i\|_2\Big)^2. \tag{5}$$

The cosine term aligns the poisoned-data gradient with the target-model fitting direction. The second term matches each poisoned target's residual magnitude around the pretrained prediction to the corresponding clean residual magnitude. The coefficient α controls this trade-off. Starting from the clean labels, we minimize L_P by projected gradient descent on the selected next-state targets. Appendix E.4 gives the projection and optimization settings.

Trajectory-consistent poisoning. The formulation above treats fine-tuning data as transition tuples. Sequential data impose an additional requirement: the successor of one transition is the current state of the next. Editing these occurrences separately would break the stored trajectory. We therefore share one editable value across both occurrences of an intermediate state, excluding trajectory boundaries. For a single edited intermediate state $\tilde{s}'_t$, with its neighboring states unchanged, the two adjacent prediction-error terms become

$$\underbrace{\|\tilde{s}'_t - P_{\psi}(s_t, a_t)\|_2^2}_{\text{transition } t} + \underbrace{\|s'_{t+1} - P_{\psi}(\tilde{s}'_t, a_{t+1})\|_2^2}_{\text{transition } t+1}.$$

Thus, the edited state influences training both as a successor target and as the next model input. With several edits, shared values are used wherever they occur, and G_{real} is the gradient of the complete rewritten fine-tuning loss, with each transition counted once. G_{target} is unchanged. Here r_p counts edited intermediate states, and neighboring edits can affect overlapping transition tuples. Appendix F.9 gives the implementation and results.

3.3 Defenses Against Data Poisoning

We evaluate stealthiness against three stages of defense: pre-training detection before poisoned transitions are used for fine-tuning, robust training during fine-tuning, and test-time monitoring.

Pre-training detection. A defender with a reference model P_{ψ_0} can compare poisoned transitions with its predictions [Chen et al., 2021]. With $p_i = P_{\psi_0}(s_i, a_i)$, we measure $\delta_d = \frac{1}{|D|}\sum_{i=1}^{|D|} \frac{\|\tilde{s}'_i - p_i\|_2}{\|p_i\|_2}, \delta_{d,\mathrm{ref}} = \frac{1}{|D|}\sum_{i=1}^{|D|} \frac{\|s'_i - p_i\|_2}{\|p_i\|_2}$. Both averages include the unchanged rows. The clean reference captures natural model prediction error. Residual-CUSUM additionally scores the sequence of unnormalized prediction residuals along each trajectory, using task-specific clean calibration (Appendix F.5.1). Appendix F.5.2 provides a low-dimensional visualization of clean and poisoned latent targets.

Training-time defense. TRIM [Jagielski et al., 2018] alternates dynamics fitting with selection of the transitions having the lowest one-step prediction losses. With trimming fraction β, each round retains the $\lfloor (1-\beta)|D| \rfloor$ transitions having the lowest losses. Matched clean-data controls measure the utility cost of applying the same procedure to benign data.

Test-time monitoring. For a trajectory $\tau = \{(s_t, a_t, s'_t)\}_{t=0}^{T-1}$ generated by a deployed model, define $\delta_m(\psi; \tau) = \frac{1}{T}\sum_{t=0}^{T-1} \frac{\|P_\psi(s_t, a_t) - s'_t\|_2}{\|s'_t\|_2}$. Each model is scored on its own executed trajectories. The clean reference $\delta_{m,\mathrm{ref}}$ evaluates P_{ψ_0} on the matched Clean model's trajectories. Each model's δ_m uses its own rollout inputs, so the comparison reflects both prediction error and the states visited. We average these trajectory-level measurements over evaluation episodes. Thus δ_d describes the adaptation dataset, while δ_m measures prediction error encountered after adaptation. Appendix D.3 gives the stochastic formulation.

4 Experiments

We evaluate SWAAP's effectiveness and stealthiness under the criteria in Section 3.3. After describing setup and metrics, we present main results on return degradation, δ_d, δ_m, and evaluated defenses, then summarize ablations and broader settings including larger datasets, trajectory-consistent poisoning, DINO-WM, and gray-box transfer. Additional implementation details, system configurations, hyperparameters, additional ablations and extensions are deferred to Appendices E–H.

4.1 Experimental Setup and Metrics

We use pretrained TD-MPC2 [Hansen et al., 2024] checkpoints on five DMControl tasks [Tassa et al., 2018], two MyoSuite tasks [Caggiano et al., 2022], and two MetaWorld tasks [Yu et al., 2020]. Each fine-tuning dataset contains $5{,}000$ transitions. At $r_p = .1$, SWAAP modifies exactly 500 next-state targets. Fine-tuning updates only the dynamics model, using Adam with a new optimizer state for each run, batch size 256, and ten epochs at task-specific learning rates chosen using clean data. MPC uses 512 candidate rollouts and planning horizon 3. SWAAP (Random) uses random transition targets and selects each transition independently with probability r_p. Direct Model Poisoning deploys the selected Stage 1 dynamics without data fine-tuning. For ManiSkill2, the same constructor uses $\alpha = .5$ on Lift and .7 on Pick and Stack; fine-tuning uses learning rates 1.75×10^{-4}, 5×10^{-5} and 2.5×10^{-4} for 25, 50 and 10 epochs, respectively. Before return-based selection, ManiSkill2 candidates satisfy $\delta_d \leq 1.25\delta_{d,\mathrm{ref}}$ and a nominal-5%-threshold alarm rate at most five percentage points above the independently measured Clean false-positive rate. Appendix E.4 gives the fine-tuning and target-selection procedure.

For each task in these three suites, we use three independently seeded Stage 1 runs satisfying the selection criteria in Appendix E.4. One target model is selected from the saved checkpoints of each run. Each resulting poisoned dataset is evaluated with three separate fine-tuning runs, using different random permutations of the training data and 100 test episodes per fine-tuned model. Clean and poisoned conditions use the same data permutations and evaluation random seeds. The test episodes are separate from those used for target selection. We average the three fine-tuning results for each Stage 1 run, then report the mean and sample standard deviation across the three Stage 1 runs. Data-level deviation δ_d is measured once per poisoned dataset over all $5{,}000$ transitions. For DMControl, MyoSuite and MetaWorld, the $\alpha = .9$ setting in Table 1 was chosen after the sensitivity

Table 1: Main attack results at $r_p = .1$. Return is cumulative test return; δ_d/δ_m are normalized data/rollout errors. Entries show mean ± sample SD across three Stage 1 runs (Lift: two); fitted methods average three final 100-episode fits per run. Clean Reference gives clean-fitted Return/$\delta_{d,\mathrm{ref}}$ and pretrained $\delta_{m,\mathrm{ref}}$ on Clean rollouts. Direct Model Poisoning Return/δ_m evaluate the target model; δ_d^* measures target-label-copying DMP-data. SWAAP and Random use $\alpha = .9$ except ManiSkill2 (Lift .5, Pick/Stack .7). Data metrics are measured once per dataset. † denotes positive SD rounded to zero. ManiSkill2 $\delta_{m,\mathrm{ref}}$ uses additional evaluations of frozen Clean checkpoints on matched seed panels; other entries retain their recorded evaluations.

Env.		Clean Reference			SWAAP (Random)			Direct Model Poisoning			SWAAP		
		Return	$\delta_{d,\mathrm{ref}}$	$\delta_{m,\mathrm{ref}}$	Return	δ_d	δ_m	Return	δ_d^*	δ_m	Return	δ_d	δ_m
DMControl	humanoid-walk	857 ± 4	$.090 \pm .000$	$.093 \pm .000^\dagger$	854 ± 5	$.090 \pm .000^\dagger$	$.094 \pm .001$	33 ± 45	$.090 \pm .005$	$.223 \pm .014$	387 ± 332	$.090 \pm .000^\dagger$	$.163 \pm .052$
	humanoid-run	595 ± 5	$.051 \pm .000$	$.049 \pm .001$	593 ± 2	$.052 \pm .000^\dagger$	$.048 \pm .000^\dagger$	2 ± 1	$.056 \pm .005$	$.222 \pm .045$	349 ± 78	$.052 \pm .000^\dagger$	$.096 \pm .011$
	dog-walk	770 ± 13	$.059 \pm .000$	$.059 \pm .001$	778 ± 14	$.059 \pm .000^\dagger$	$.060 \pm .001$	351 ± 151	$.063 \pm .002$	$.125 \pm .020$	725 ± 58	$.060 \pm .000^\dagger$	$.067 \pm .008$
	dog-run	$648 \pm 0^\dagger$	$.051 \pm .000$	$.050 \pm .000^\dagger$	647 ± 2	$.051 \pm .000^\dagger$	$.049 \pm .000^\dagger$	62 ± 61	$.055 \pm .004$	$.125 \pm .034$	507 ± 124	$.052 \pm .000^\dagger$	$.062 \pm .009$
	cheetah-run	835 ± 1	$.028 \pm .000$	$.030 \pm .000^\dagger$	834 ± 2	$.028 \pm .000^\dagger$	$.025 \pm .000^\dagger$	475 ± 177	$.031 \pm .001$	$.087 \pm .027$	787 ± 42	$.030 \pm .000^\dagger$	$.035 \pm .007$
MyoSuite	pen-twirl-hard	3350 ± 70	$.056 \pm .000$	$.057 \pm .001$	3340 ± 73	$.056 \pm .000^\dagger$	$.046 \pm .001$	1782 ± 486	$.063 \pm .003$	$.104 \pm .008$	3285 ± 118	$.056 \pm .000^\dagger$	$.050 \pm .002$
	reach-hard	749 ± 11	$.035 \pm .000$	$.061 \pm .002$	749 ± 12	$.035 \pm .000^\dagger$	$.030 \pm .000^\dagger$	-925 ± 2418	$.040 \pm .002$	$.161 \pm .119$	679 ± 73	$.035 \pm .000^\dagger$	$.088 \pm .047$
Meta-World	push	1764 ± 12	$.080 \pm .000$	$.082 \pm .001$	1754 ± 7	$.080 \pm .000^\dagger$	$.085 \pm .002$	1082 ± 748	$.086 \pm .002$	$.194 \pm .096$	1409 ± 275	$.081 \pm .000^\dagger$	$.137 \pm .036$
	soccer	1720 ± 5	$.046 \pm .000$	$.042 \pm .001$	1721 ± 3	$.046 \pm .000^\dagger$	$.031 \pm .000^\dagger$	1090 ± 613	$.046 \pm .008$	$.132 \pm .079$	1147 ± 306	$.046 \pm .000^\dagger$	$.114 \pm .040$
ManiSkill2	lift-cube	$193 \pm 0^\dagger$	$.018 \pm .000^\dagger$	$.021 \pm .000^\dagger$	$193 \pm 0^\dagger$	$.026 \pm .000^\dagger$	$.023 \pm .000^\dagger$	128 ± 79	$.029 \pm .009$	$.164 \pm .116$	192 ± 1	$.022 \pm .000^\dagger$	$.028 \pm .002$
	pick-cube	160 ± 2	$.026 \pm .000^\dagger$	$.028 \pm .002$	161 ± 1	$.031 \pm .001$	$.026 \pm .002$	133 ± 18	$.038 \pm .005$	$.157 \pm .053$	159 ± 2	$.030 \pm .001$	$.034 \pm .003$
	stack-cube	174 ± 5	$.042 \pm .000^\dagger$	$.043 \pm .003$	172 ± 3	$.047 \pm .000^\dagger$	$.045 \pm .003$	114 ± 43	$.042 \pm .004$	$.118 \pm .038$	156 ± 13	$.043 \pm .000^\dagger$	$.057 \pm .010$

experiments, with the selected targets kept unchanged. Appendix F.2 shows the variation across Stage 1 and fine-tuning runs separately.

4.2 Effective and Stealthy Poisoning

Main attack results. Table 1 evaluates the complete attack after poisoned fine-tuning. SWAAP lowers return relative to the listed clean references while modifying only a bounded fraction of transition targets and keeping δ_d close to $\delta_{d,\mathrm{ref}}$ in most tasks. The resulting δ_m varies across environments, so we do not claim model-level indistinguishability; rather, SWAAP demonstrates an effectiveness–stealth trade-off under bounded data-level perturbation. The across-Stage 1-run SDs show varied attack strength. With $r_p = .1$ and $\alpha = .9$, SWAAP reduces mean return on all nine DMControl, MyoSuite and MetaWorld tasks while keeping data deviations close to the clean reference. The reductions reach 54.7% on Humanoid-Walk, 41.4% on Humanoid-Run, 33.3% on MW-Soccer, 21.7% on Dog-Run and 20.2% on MW-Push. The other four reductions range from 2.0% to 9.5%. SWAAP (Random) remains close to Clean in these tasks. The difference supports using an intentionally harmful target to direct the fine-tuning updates. The ManiSkill2 reductions are 0.60%, 0.84% and 10.20% on Lift, Pick and Stack. Together, the results show that bounded changes to fine-tuning targets can alter control performance even when their average prediction residual remains near that of clean data. Figure 2 compares stealth weights, and Appendix F.1 gives the complete sensitivity results.

Comparison with PGD poisoning. The offline PGD baseline modifies successor targets to reduce a pretrained policy's value directly. SWAAP instead optimizes how those targets affect world-model fine-tuning. Table 8 compares the two approaches on MW-Push. Increasing PGD's perturbation bound from .015 to .05 changes Return from 1776 to 1736 while increasing δ_d from .0829 to .0984. SWAAP reaches Return 1409 with $\delta_d = .0807$, close to the Clean reference of .0799. Thus, in this comparison, optimizing the training effect produces greater degradation at lower measured data deviation than directly lowering the value of individual successor states. Appendix F.8 gives the baseline objective and the separate online state-perturbation comparison.

Pre-training detection. We evaluate residual-CUSUM [Page, 1954] on poisoned fine-tuning trajectories using task-specific clean calibration. Table 5 reports full AUC, standardized low-FPR partial AUC, and alarm rates at nominal 5% and 10% thresholds. At $\alpha = .1$, mean TPRs at these thresholds reach $72\%/84\%$ on MW-Push and $42\%/50.7\%$ on Myo-Pen-Twirl-Hard. At $\alpha = .9$, the mean AUC across the nine tasks is $.506$, and each task's 5%-threshold alarm rate equals that of its corresponding unmodified source dataset. Humanoid-Run's 20% rate, for example, is already present in two of its ten clean source trajectories. Realized FPRs on the separate clean test cohorts and the full detector results are given in Appendix F.5.1.

Training-time defense. We use $\beta = .2$ at $r_p = .1$, retaining 4,000 of 5,000 transitions per trimming round. TRIM improves mean poisoned returns on Humanoid-Walk and Dog-Run with small mean changes in clean utility (Table 7). On Humanoid-Walk, poisoned return increases from 387.0 to 483.8, while Clean return changes from 856.9 to 857.6. A substantial clean-to-poison gap remains. On Dog-Run, poisoned return rises from 507.5 to 613.3. Myo-Reach-Hard exhibits a different trade-off:

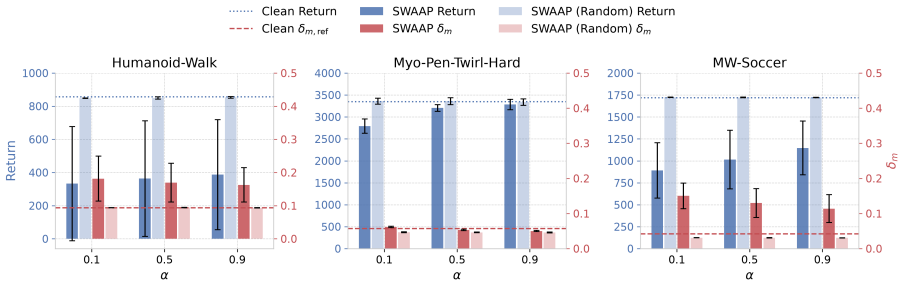
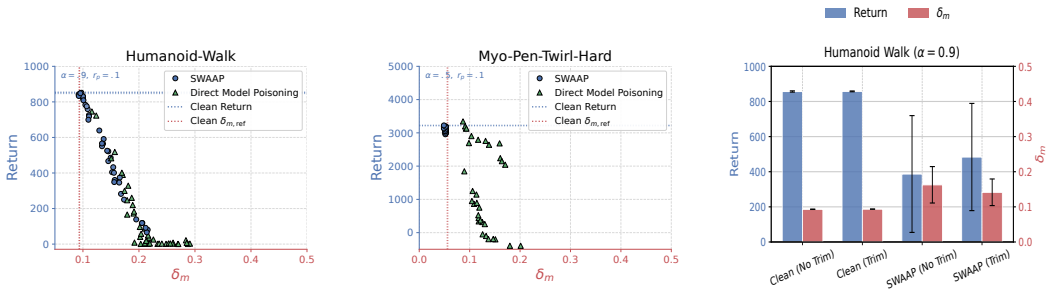


Figure 2: Return (blue) and rollout error δ_m (red) for SWAAP (dark) and SWAAP (Random) (light) across α at $r_p = .1$. Dotted blue and dashed red lines denote matched Clean Return and pretrained-model error on Clean rollouts. Targets and supports are fixed across α. Error bars are sample SDs across three Stage 1 runs, each averaging three fits of 100 episodes.



(a) Humanoid-Walk (b) Myo-Pen-Twirl-Hard (c) TRIM defense

Figure 3: (a,b) Return versus rollout error δ_m: Direct target models (green triangles) and SWAAP from the same targets (blue circles). We show 43 pairs from three Humanoid-Walk Stage 1 runs and 33 from two Myo-Pen-Twirl-Hard runs, with $(r_p, \alpha) = (.1, .9)$ and $(.1, .5)$, respectively. Direct uses ten episodes; SWAAP uses one development fit and 100 episodes. Dotted lines mark Clean references. (c) Humanoid-Walk TRIM at $(r_p, \alpha, \beta) = (.1, .9, .2)$: bars show Return (blue) and δ_m (red); error bars are sample SD across three Stage 1-run means, each averaging three 100-episode fits.

TRIM reduces poisoned return from 678.8 to 633.4 and Clean return from 749.2 to 618.7. Figure 3c illustrates Humanoid-Walk result. Appendix F.7 reports the paired effects and fitting budgets.

Stage-wise realization and attack-stealth trade-off. Figure 2 examines Stage 2 across stealth weights α and shows that, at relaxed stealth, SWAAP's return drop substantially exceeds that of SWAAP (Random). Appendix F.3 further checks this comparison when both methods modify the same transitions. Additional α and r_p ablations across all environments are in Appendix F.1. Figure 3 visualizes the attack-stealth frontier induced by different Stage 1 checkpoints and Stage 2 poisoning settings. In Stage 1, optimization produces saved target-model checkpoints with different return-δ_m trade-offs; Direct Model Poisoning evaluates these checkpoints directly, while SWAAP attempts to realize selected targets through constrained data poisoning. The comparison exposes the main realization bottleneck. In Humanoid-Walk, SWAAP points closely track the Direct Model Poisoning frontier, showing that Stage 2 can recover much of the target model's effect. In Myo-Pen-Twirl-Hard, however, SWAAP remains separated from the Direct Model Poisoning frontier, showing that harmful targets are not always equally realizable through gradient-matched data poisoning. This target-selection procedure reflects a limitation of the current two-stage design that is not jointly optimized: Stage 1 optimizes for harmful and low-δ_m target dynamics, but not for how easily those targets can later be realized by Stage 2.

Trajectory-consistent poisoning. The main attack edits successor labels in transition data. For sequential fine-tuning, we instead share each edited intermediate state across the two transitions in which it occurs. This preserves the recorded trajectory's state continuity and makes the poison influence both the preceding prediction target and the following model input. Editing 250 intermediate states yields Return reductions of 10.35%–95.12% across three Humanoid-Walk targets and 5.82%–25.38% across two Dog-Run targets. The corresponding data deviations are higher than Clean, as

reported in Appendix F.9. The experiment demonstrates that data poisoning can retain its effect while satisfying this sequential consistency requirement.

4.3 Ablations and Extensions

Stage 1 target realization. The comparison with Direct Model Poisoning separates the ability to identify harmful dynamics from the ability to induce them through training data. Direct installation lowers mean return on all nine DMControl, MyoSuite and MetaWorld tasks. SWAAP approaches the effects of some Humanoid-Walk targets, while the larger gaps on Dog-Walk, Cheetah-Run and MyoSuite identify target realization as the next point to examine. We thus compare SWAAP with DMP-data, which directly copies target predictions into the same selected transitions. Under the matched-δ_d, gradient matching gives greater mean return reduction on four of nine tasks, including MW-Push (27.49% versus 7.31%) and MW-Soccer (11.17% versus 2.74%). This comparison shows that the way labels shape the training update matters even at the same mean data deviation. The two-stage decomposition makes target harmfulness and data realization observable separately, helping explain the range of final attack strengths. Appendix F.6 relates these results to the different roles of target harmfulness, editable transitions, and fine-tuning dynamics.

DINO-WM. To examine a different world-model representation, we adapt SWAAP to DINO-WM's visual features and history-dependent predictions. At $r_p = .1$, poisoned fine-tuning reduces test success by 8.33 ± 2.89 and 6.67 ± 2.89 percentage points for $\alpha = 0$ and .9, respectively, across three fine-tuning runs on the same 20 test goals. Both poisoned datasets produce the same CUSUM alarm counts as their unmodified versions. This establishes a data-poisoning effect in a visual world-model pipeline with a different encoder and planner. Appendix H.1 reports the shared target and data, representation-specific construction, and the separate success and state-distance measurements.

Gray-box transfer. Finally, we relax access to the victim model by constructing raw-observation poisons with a single-task Soccer surrogate and supplying them to a multi-task TD-MPC2 victim with its own encoder. Soccer Return falls by $9.08\%/5.20\%$ and Push by $10.78\%/17.84\%$ at the two victim learning rates. Across 80 tasks, at least 5% degradation occurs on 16/12 tasks, but median reductions are $-0.32\%/-0.41\%$ and other tasks improve. The victim-reference detector has AUC .639 and $2\%/12\%$ alarm rates at nominal $5\%/10\%$ thresholds. These results demonstrate cross-encoder transfer with selective effects (Appendix H.2).

Transition Selection, planning configurations, poisoning budget and larger finetuning size. We further measure how different settings in the pipeline affect the poison effect. Replacing the top-r_p transition-selection heuristic with random selection weakens the attack on Humanoid-Walk, supporting the choice of poisoning transitions most inconsistent with the Stage 1 target. SWAAP lowers mean Humanoid-Walk return in all fifteen tested planner configurations. Raising r_p from .01 to .1 strengthens the attack; returns at .2 and .3 are similar for fixed targets. Enlarging the dataset dilutes a fixed edit budget, whereas holding $r_p = .1$ retains substantial degradation at 25,000 transitions on Humanoid-Walk and Dog-Run (Appendices G.1–G.4).

5 Conclusion and Limitations

We introduced SWAAP, a two-stage poisoning framework that uses stealth-constrained gradient matching to manipulate world-model fine-tuning. Our evaluations expose vulnerabilities in world-model adaptation and motivate stronger protection of training data and learned dynamics.

As an initial study of world model poisoning, this work has limitations. First, our experiments primarily use TD-MPC2 as the victim model, with DINO-WM as an additional backbone. Scaling to larger foundation models is left for future work. The main construction also assumes white-box model access and visibility of the entire fine-tuning dataset. Partial visibility requires an additional approximation to the hidden data. Second, SWAAP uses a two-stage approximation: Stage 1 identifies harmful, low-deviation target dynamics, and Stage 2 seeks to induce their effects via gradient-matched data poisoning. Since some targets are easier to realize than others, jointly optimizing target selection and data realization is an important next step. Third, we focus on untargeted performance degradation. Targeted behavior manipulation remains a separate problem. The evaluated defenses use only clean calibration, prediction errors and training losses. Further evaluation could examine defenses informed by the attack target or gradient-matching objective, deployment-specific safeguards and stricter poisoning constraints.

Acknowledgments

This work has been funded in part by NSF grant CNS-2146548. We thank the anonymous reviewers for their insightful feedback.

References

Jonathan F Bard. *Practical bilevel optimization: algorithms and applications*, volume 30. Springer Science & Business Media, 2013.

Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *International Conference on Machine Learning (ICML)*, 2012.

Jake Bruce, Michael Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Bechtle, Feryal Behbahani, Stephanie Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. Genie: Generative interactive environments, 2024. URL `https://arxiv.org/abs/2402.15391`.

Vittorio Caggiano, Huawei Wang, Guillaume Durandau, Massimo Sartori, and Vikash Kumar. MyoSuite: A contact-rich simulation suite for musculoskeletal motor control. In *Annual Learning for Dynamics and Control Conference*, 2022.

Eduardo F Camacho and Carlos Bordons. *Model predictive control*. Springer, 2013.

Jian Chen, Xuxin Zhang, Rui Zhang, Chen Wang, and Ling Liu. De-pois: An attack-agnostic defense against data poisoning attacks. *IEEE Transactions on Information Forensics and Security*, 16: 3412–3425, 2021.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, 2011.

Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Security Symposium*, 2020.

Jonas Geiping, Liam Fowl, W. Ronny Huang, Wojciech Czaja, Gavin Taylor, Michael Moeller, and Tom Goldstein. Witches' brew: Industrial scale data poisoning via gradient matching. In *International Conference on Learning Representations*, 2021.

Saeed Ghadimi and Mengdi Wang. Approximation methods for bilevel programming. *arXiv preprint arXiv:1802.02246*, 2018.

Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. Maniskill2: A unified benchmark for generalizable manipulation skills, 2023. URL `https://arxiv.org/abs/2302.04659`.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

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 (ICML)*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *International Conference on Machine Learning (ICML)*, 2019.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024. URL `https://arxiv.org/abs/2301.04104`.

Nicklas Hansen, Hao Su, and Xiaolong Wang. TD-MPC2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations (ICLR)*, 2024.

W. Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. Metapoison: Practical general-purpose clean-label data poisoning. In *Advances in Neural Information Processing Systems*, 2020.

Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *IEEE Symposium on Security and Privacy (SP)*, 2018.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning (ICML)*, 2022.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. In *International Conference on Learning Representations (ICLR)*, 2020.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2016.

Bo Liu, Mao Ye, Stephen Wright, Peter Stone, and Qiang Liu. BOME! bilevel optimization made easy: A simple first-order approach. In *Advances in neural information processing systems (NeurIPS)*, 2022.

Risheng Liu, Jiaxin Gao, Jin Zhang, Deyu Meng, and Zhouchen Lin. Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):10045–10067, 2021.

Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *arXiv preprint arXiv:1903.03088*, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *10th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2017.

Nvidia. Cosmos world foundation model platform for Physical AI. `https://arxiv.org/abs/2501.03575`, 2025a.

Nvidia. World foundation models. `https://www.nvidia.com/en-us/glossary/world-models/`, 2025b.

OpenAI. Creating video from text. `https://openai.com/index/sora/`, 2025.

E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954. ISSN 00063444. URL `http://www.jstor.org/stable/2333009`.

Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, and Sam Devlin. Imitating human behaviour with diffusion models. In *International Conference on Learning Representations (ICLR)*, 2023.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning (ICML)*, 2016.

Amin Rakhsha, Goran Radanovic, Rati Devidze, Xiaojin Zhu, and Adish Singla. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2020.

Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

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 (CoRL)*, 2020.

Xuezhou Zhang, Yuzhe Ma, Adish Singla, and Xiaojin Zhu. Adaptive reward-poisoning attacks against reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2020.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. DINO-WM: World models on pre-trained visual features enable zero-shot planning, 2025. URL `https://arxiv.org/abs/2411.04983`.

Appendix Table of Contents

A Use of LLMs **16**

B Broader Impact **16**

C Related Works **16**

C.1 World Models 16

C.2 Model-Based Reinforcement Learning 17

C.3 Model Poisoning Attack 17

C.4 Data Poisoning Attacks and Defenses 17

D Theory and Derivations **18**

D.1 Derivation of Transition Gradient 18

D.2 BOME Details for Stage 1 Model Poisoning 19

D.3 Stochastic Transition Functions Case 21

E Implementation Details **21**

E.1 Model Predictive Control 22

E.2 Hyperparameters 22

E.3 Stage 1: Target Model Identification 23

E.4 Stage 2: Data Poisoning 24

E.5 Hardware Specification and Computational Cost 25

F Main Extended Results **26**

F.1 SWAAP and SWAAP (Random) 26

F.2 Repeated Fine-Tuning Experiments 26

F.3 Transition Selection in SWAAP (Random) 26

F.4 Model Poisoning Training Curve 28

F.5 Pre-Training Detection 28

F.5.1 Residual-Based CUSUM Pre-Training Detector 29

F.5.2 Latent-Space PCA Visualization 30

F.6 Stage 1–Stage 2 Realization Bottleneck 31

F.7 TRIM Defense results 32

F.8 Iterative FGSM/PGD Transition-Perturbation Baselines 33

F.9 Trajectory-Consistent Data Poisoning 33

G Additional Ablations **34**

G.1 Top-r_p Transition Selection vs. Random Selection 34

G.2 MPC Hyperparameter Ablation 34

G.3 Stage 2 Poisoning Ratio r_p 35

G.4 Larger Fine-Tuning Datasets 35

H Extensions **36**

H.1 DINO-WM Push-T Results 36

H.2 Gray-Box Attack on TD-MPC2 Multi-Task World Model 37

Appendix

A Use of LLMs

The authors used LLM-based assistants and coding agents for manuscript preparation, supplementary-material presentation, and software-release support. Their uses included grammar correction, wording suggestions, condensation of prose, LaTeX/formatting assistance, improving the organization and readability of author-written code artifacts, organizing the public code repository, preparing documentation and release packages, and assisting with verification and publication of the GitHub code release. All algorithms, experiments, figures, tables, citations, and technical claims were produced, checked, and approved by the authors. No confidential reviewer information, proprietary data, or sensitive personal data were used as LLM inputs.

B Broader Impact

This paper studies a new training-time attack surface for learned world models. The primary intended benefit is defensive: by showing that fine-tuning trajectories can manipulate learned dynamics and downstream planning, the work motivates stronger data validation, robust fine-tuning, and monitoring methods for world-model adaptation pipelines. These issues are especially important for embodied agents, robotics, autonomous systems, and other settings where world models may be updated from locally collected experience.

The same techniques could also inform misuse by suggesting ways to craft poisoned fine-tuning data. We mitigate this risk by studying the attack in controlled simulation benchmarks, limiting the threat model to bounded transition-target poisoning, and evaluating representative detection and robust-training defenses. We do not provide instructions for attacking deployed real-world systems. We believe that disclosure is warranted because the vulnerability arises naturally in continual fine-tuning pipelines, and understanding it is necessary for developing effective defenses.

More broadly, this work highlights that the robustness of world-model agents cannot be evaluated only at the final policy or planner level. The training data stream, learned dynamics, and downstream planning procedure are coupled, so future safety evaluations should consider how small changes in adaptation data can propagate into long-horizon behavior. We hope the proposed evaluations encourage further work on secure data collection, provenance tracking, anomaly detection, and robust world-model fine-tuning.

C Related Works

C.1 World Models

World models aim to learn compact and predictive representations of environment dynamics that can be used for planning and control. Instead of interacting directly with the environment, an agent can rely on its learned model to simulate trajectories, evaluate policies, and anticipate future states. First introduced in [Ha and Schmidhuber, 2018], this work introduced a three-component architecture combining a variational autoencoder (VAE), a recurrent neural network (RNN), and a linear controller, demonstrating that policies trained entirely within a latent model can transfer back to the real environment. This work sparked a line of research exploring increasingly powerful and scalable model-based reinforcement learning frameworks.

Dreamer-style agents use latent dynamics models optimized end-to-end with reinforcement learning. DreamerV3 [Hafner et al., 2024] achieves state-of-the-art performance on visual control and robotic tasks. It employs a recurrent state-space model (RSSM) with deterministic state h_t and stochastic latent s_t, performing rollouts entirely in latent space. Three heads are trained on the RSSM: a reward predictor $\hat{r}_t$, a value function $\hat{V}(\cdot)$, and an actor π, using imagined rollouts. At decision time, the actor proposes candidate actions, either sampled or taken as the mean.

More recent approaches have investigated architectural advances tailored for large-scale and complex environments. DINO-WM [Zhou et al., 2025] leverages self-supervised vision transformers (ViTs) to improve perception quality, enabling stronger generalization in visually rich settings. Similarly,

TD-MPC2 [Hansen et al., 2024] proposes a temporally abstracted model predictive control framework that combines world models with trajectory optimization, achieving sample-efficient learning and strong performance in high-dimensional continuous control tasks.

Parallel to these developments, diffusion-based world models have emerged as a promising alternative [Pearce et al., 2023, Janner et al., 2022]. By parameterizing the transition distribution as a denoising diffusion process, these models can capture multi-modal and stochastic dynamics more effectively than conventional Gaussian latent models. Diffusion world models have been shown to improve both planning quality and robustness to uncertainty, making them attractive in settings where dynamics are highly non-deterministic.

C.2 Model-Based Reinforcement Learning

Model-based reinforcement learning (MBRL) is a powerful paradigm that improves sample efficiency and enables better generalization compared to purely model-free methods. In MBRL, an agent learns an explicit dynamics model of the environment and leverages this model for planning or policy optimization. We study an agent that leverages a learned world model to interact with an environment formalized as a Markov decision process (MDP) $(S, A, P, R, \gamma, \mu)$, where μ is the initial state distribution, S is the state space, A is the action space, $P : S \times A \to \Delta(S)$ is the transition kernel, $R : S \times A \times S \to \mathbb{R}$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor. The goal is to find a policy $\pi : S \to \Delta(A)$ that maximizes the expected discounted return $\sum_{i=0}^{T} \gamma^i R(s_t, a_t)$.

Unlike model-free approaches that learn value functions or policies directly from experience [Mnih et al., 2015, Lillicrap et al., 2016, Haarnoja et al., 2018], MBRL explicitly learns a parametric approximation $\hat{P}_\psi$ of the transition kernel P, often using neural networks [Deisenroth and Rasmussen, 2011, Chua et al., 2018, Janner et al., 2019, Hansen et al., 2024]. This learned dynamics model can then be used for model-predictive control (MPC) [Camacho and Bordons, 2013], trajectory optimization [Tassa et al., 2012], or to generate synthetic rollouts for policy improvement [Kaiser et al., 2020, Hafner et al., 2019].

C.3 Model Poisoning Attack

Model poisoning attacks directly compromise a learned model by altering its parameters. They assume the adversary can craft malicious model updates to steer the learning outcome or directly replace model parameters. For example, carefully crafted local malicious updates can harm the performance of a federated learning system [Fang et al., 2020].

In reinforcement learning, model poisoning is particularly concerning when applied to the agent's learned dynamics model. Small, adversarial modifications to the transition kernel can propagate over planning horizons, misleading policy improvement, and causing degraded performance. This makes model poisoning a uniquely severe threat in model-based RL, as even subtle deviations from the true dynamics can cascade into large errors in long-term decision making.

C.4 Data Poisoning Attacks and Defenses

Data poisoning attacks compromise learning systems by corrupting the training dataset to bias the learned model toward an adversarial objective. Unlike model poisoning, which manipulates parameters directly, data poisoning assumes the attacker can only influence the data stream but not the learning algorithm itself, which is a more realistic attack scenario. Data poisoning attacks have been shown to significantly degrade the model performance even with small amounts of poisoned data [Biggio et al., 2012]. A recent and effective data poisoning technique leverages gradient matching, which optimizes poisoned samples such that their gradients closely align with those of a target adversarial objective [Geiping et al., 2021]. By ensuring that poisoned data induces updates similar to those of the adversary's intended solution, gradient matching enables stealthy and highly effective poisoning even under limited attacker control. We also adopt gradient matching in the second stage of our attack methods, where we inject carefully crafted poisoned transitions into the newly collected fine-tuning dataset.

In reinforcement learning, data poisoning is particularly dangerous since training relies on sequential interactions with the environment. By injecting corrupted transitions into the finetune data or

modifying observed trajectories, an adversary can degrade the long-term performance of the agent, or even embed targeted failures [Rakhsha et al., 2020, Zhang et al., 2020].

Effective defenses against data poisoning attacks involve detection and training-time strategies. We consider two widely used data poisoning defenses in our work. Chen et al. [2021] propose De-Pois, an attack-agnostic detection method that identifies poisoned data points with the help of a mimic model trained from clean data samples. By measuring the difference between the samples on the mimic model's outputs, De-Pois can flag and remove suspicious points, improving robustness without assuming knowledge of the attack type.

Complementing detection, training-time defenses aim to mitigate the effect of poisoned samples during learning. For example, [Jagielski et al., 2018] introduce the TRIM method and its iterative variant, which estimates and removes a fraction of potentially poisoned points based on deviation and the statistical properties of the data.

D Theory and Derivations

D.1 Derivation of Transition Gradient

For a fixed policy π_θ, the discounted infinite-horizon counterpart of Theorem 3.1 is

$$\nabla_\psi J(P_\psi, \pi_\theta) = \frac{1}{1-\gamma}\,\mathbb{E}_{d_\pi, \pi_\theta, P_\psi}\Big[(R(s,a,s') + \gamma V(s'))\nabla_\psi \log P_\psi(s' \mid s, a)\Big],$$

where $0 \leq \gamma < 1$ and d_π is the normalized discounted state-visitation distribution under P_ψ, π_θ and the initial distribution. Actions and successors are sampled from π_θ and P_ψ. The policy parameters, reward function and initial distribution are held fixed when differentiating with respect to ψ. We assume the expectations are finite and differentiation can be interchanged with summation or integration. **Proof:** We start from the derivative of the state-value function of an arbitrary initial state s_0 and extend the state into the future time indefinitely, and expand it to write it as a recursive form in terms of M function defined below, from which it unrolls to become an infinite series of the sum of $M(s_k)$ weighted by the probability of reaching s_k from s_0 in k steps.

$$\begin{aligned}
\nabla_\psi V(s_0) &= \nabla_\psi \sum_{a_0} \pi(a_0|s_0) Q(s_0, a_0) \\
&= \nabla_\psi \sum_{a_0} \pi(a_0|s_0) \sum_{s_1} P_\psi(s_1|s_0,a_0)[R(s_0,a_0,s_1) + \gamma V(s_1)] \\
&= \sum_{a_0} \pi(a_0|s_0) \sum_{s_1} \nabla_\psi P_\psi(s_1|s_0,a_0)[R(s_0,a_0,s_1) + \gamma V(s_1)] \\
&\quad + \gamma \sum_{a_0} \pi(a_0|s_0) \sum_{s_1} P_\psi(s_1|s_0,a_0) \nabla_\psi V(s_1) \\
&= M(s_0) + \gamma \sum_{s_1} \rho_\pi(s_0 \to s_1, k{=}1) \nabla_\psi V(s_1) \\
&= M(s_0) + \gamma \sum_{s_1} \rho_\pi(s_0 \to s_1, 1)\Big[M(s_1) + \gamma \sum_{s_2} \rho_\pi(s_1 \to s_2, 1) \nabla_\psi V(s_2)\Big] \\
&= M(s_0) + \sum_{s_1} \gamma \rho_\pi(s_0 \to s_1, 1) M(s_1) + \sum_{s_2} \gamma^2 \rho_\pi(s_0 \to s_2, 2) M(s_2) + \cdots \\
&= \sum_{x} \sum_{k=0}^{\infty} \gamma^k \rho_\pi(s_0 \to x, k) M(x) \\
&= \sum_{s} \eta(s) M(s) \\
&= \frac{1}{1-\gamma} \sum_{s} \frac{\eta(s)}{\sum_x \eta(x)} M(s) \\
&= \frac{1}{1-\gamma} \sum_{s} d_\pi(s) \sum_{a} \pi(a|s) \sum_{s'} \nabla_\psi P_\psi(s'|s,a)(R(s,a,s') + \gamma V(s')).
\end{aligned}$$

where $M(s_0) \coloneqq \sum_{a_0} \pi(a_0|s_0) \sum_{s_1} \nabla_\psi P_\psi(s_1|s_0,a_0)[R(s_0,a_0,s_1) + \gamma V(s_1)]$, and $\sum_s \eta(s) = 1/(1-\gamma)$. This constant can be absorbed into an optimization step size, but is retained in the gradient identity. Here $\rho_\pi(s \to x, k=1) \coloneqq \sum_a \pi(a|s) P_\psi(x|s,a)$ is the transition probability of reaching x from s at step $k=1$, and the relation $\rho^\pi(s \to x, k+1) = \sum_{s'} \rho^\pi(s \to s', k)\rho^\pi(s' \to x, 1)$ is used to write the transition gradient into recursive form. Here d_π is conditional on s_0; averaging it over the fixed initial distribution gives the distribution used below. The log-derivative identity gives

$$\begin{aligned}\nabla_\psi J(P_\psi, \pi_\theta) &= \frac{1}{1-\gamma} \sum_s d_\pi(s) \sum_a \pi_\theta(a \mid s) \sum_{s'} \nabla_\psi P_\psi(s' \mid s,a)\big(R(s,a,s') + \gamma V(s')\big) \\ &= \frac{1}{1-\gamma} \mathbb{E}_{d_\pi, \pi_\theta, P_\psi}\Big[\big(R(s,a,s') + \gamma V(s')\big)\nabla_\psi \log P_\psi(s' \mid s,a)\Big].\end{aligned}$$

For planning with a finite horizon T, given a time-indexed value function $V_t(s)$ with terminal value $V_T(s) = 0$, the unrolling stops after $T-t$ steps. Define

$$M_t(s) = \sum_a \pi_\theta(a \mid s) \sum_{s'} \nabla_\psi P_\psi(s' \mid s,a)\big(R(s,a,s') + \gamma V_{t+1}(s')\big).$$

The corresponding recursion is

$$\begin{aligned}\nabla_\psi V_t(s) &= M_t(s) + \gamma \sum_{s'} \rho_\pi(s \to s', 1)\nabla_\psi V_{t+1}(s') \\ &= \sum_{i=0}^{T-t-1} \gamma^i \sum_x \rho_\pi(s \to x, i) M_{t+i}(x).\end{aligned}$$

The terminal gradient vanishes because $V_T = 0$. Thus, averaging over the fixed initial distribution μ, the finite-horizon transition gradient is

$$\begin{aligned}\nabla_\psi J_T(P_\psi, \pi_\theta) &= \mathbb{E}_{s_0 \sim \mu}\big[\nabla_\psi V_0(s_0)\big] \\ &= \mathbb{E}_{P_\psi, \pi_\theta}\Big[\sum_{t=0}^{T-1} \gamma^t \big(R(s_t,a_t,s_{t+1}) + \gamma V_{t+1}(s_{t+1})\big)\nabla_\psi \log P_\psi(s_{t+1} \mid s_t, a_t)\Big].\end{aligned}$$

For deterministic transitions $s_{t+1} = f_\psi(s_t, a_t)$, assume that the displayed parameter and state derivatives of f_ψ, R and V_t exist and that differentiation can be interchanged with expectation. The Jacobian $\nabla_\psi f_\psi(s,a)$ is evaluated at fixed (s,a). The same finite-horizon recursion gives

$$\begin{aligned}\nabla_\psi J_T(P_\psi, \pi_\theta) = \mathbb{E}_{f_\psi, \pi_\theta}\Bigg[&\sum_{t=0}^{T-1} \gamma^t \big(\nabla_\psi f_\psi(s_t,a_t)\big)^\top \\ &\nabla_{s'}\big(R(s_t,a_t,s') + \gamma V_{t+1}(s')\big)\big|_{s'=f_\psi(s_t,a_t)}\Bigg].\end{aligned}$$

D.2 BOME Details for Stage 1 Model Poisoning

The adversary minimizes the return of the surrogate policy that learns from perturbed transitions by solving the following bilevel optimization problem:

$$\begin{array}{ll}\min_\psi & J(P, \pi_\theta(\psi)) + \lambda L(P_\psi, P) \\ \text{s.t.} & \theta(\psi) \in \arg\max_{\theta'} J(P_\psi, \pi_{\theta'}).\end{array} \tag{6}$$

The goal is to minimize the constraint-regularized expected return with respect to the optimal policy derived from the poisoned environment.

This bilevel optimization problem remains challenging due to its nested structure. The primary difficulty lies in computing the derivative $\nabla_\psi \theta(\psi)$. If one directly treats $\theta(\psi)$ as a function of ψ, gradient-based methods such as hypergradient descent [Bard, 2013] update ψ using $\nabla_\psi J(P, \pi_\theta) = \nabla_\psi \theta(\psi) \nabla_\theta J(P, \pi_\theta)$, but evaluating $\nabla_\psi \theta(\psi) = \nabla_\psi \arg\max_{\theta'} J(P_\psi, \pi_{\theta'})$ requires solving linear systems and computing costly second-order Hessians. Prior work has attempted

Algorithm 1 Algorithm for Perturbed Model Identification

Input: env. dynamics P, dynamics model ψ_0, policy θ_0, reward model R, value net V_ϕ
Output: perturbed model $\hat{\psi}$
Initialize (ψ_k, θ_k) from (ψ_0, θ_0); keep the pretrained encoder, reward and value networks fixed
for each prescribed outer iteration k **do**
    Collect model rollouts from $(P_{\psi_k}, \pi_{\theta_k})$ to form $\mathcal{B}_{kk}$
    $\theta_k^0 \leftarrow \theta_k$
    for $i \leftarrow 0$ to $W-1$ **do**
        Sample $\mathcal{B}_{kk}$ and update θ_k^{i+1} by clipped Adam ascent using the value-based policy-gradient estimate
    Collect model rollouts from $(P_{\psi_k}, \pi_{\theta_k^W})$ to form $\mathcal{B}_{kW}$
    Collect environment rollouts with policy θ_k and planning model ψ_0 to form $\mathcal{B}_{pk}$
    Collect clean environment transitions using (ψ_0, θ_0) for the consistency loss
    $\Omega \leftarrow [\,]$
    for $i \leftarrow 1$ to N **do**
        Estimate the outer direction from $\mathcal{B}_{pk}$ and the clean consistency samples
        Estimate the constraint direction from $\mathcal{B}_{kW}$ and $\mathcal{B}_{kk}$, using θ_k for next-state actions
        Compose the sampled value-based directions and λ_k as detailed in Appendix E.3
        Append the resulting joint dynamics/policy direction to Ω
    $\omega_k \leftarrow \frac{1}{N}\sum_{\omega\in\Omega}\omega$
    Apply separate clipped Adam descent steps to ψ_k and θ_k using the components of ω_k
    Periodically save and evaluate the dynamics checkpoint
return $\hat{\psi}$ selected from saved checkpoints as described in Appendix E.4

to bypass this difficulty either through surrogate approximations or heuristics that are difficult to control [Pedregosa, 2016, Ghadimi and Wang, 2018, MacKay et al., 2019]. Moreover, reinforcement learning objectives are highly non-convex, involve high-dimensional state–action spaces, and present a complex optimization landscape. As a result, naive bilevel optimization methods often fail to converge reliably, making them unsuitable for Problem 6 [Liu et al., 2021].

To obtain a more scalable solution, we adopt the first-order dynamic barrier gradient descent method (BOME) in [Liu et al., 2022], which reformulates the problem by replacing the implicit argmin operator with a value-function constraint.

This allows the outer objective and inner objective to be solved without explicitly computing the derivative $\nabla_\psi \theta(\psi)$. Formally, let $f(\psi, \theta) := J(P, \pi_\theta) + \lambda L(P_\psi, P)$ denote the outer objective, and $q(\psi, \theta) := \max_{\theta'} J(P_\psi, \pi_{\theta'}) - J(P_\psi, \pi_\theta)$, which measures the suboptimality of π_θ relative to the optimal policy under P_ψ. Under the value-function approach, the bilevel problem becomes the following constrained optimization:

$$\min_{\psi,\theta} f(\psi, \theta) \quad \text{s.t.} \quad q(\psi, \theta) \le 0.$$

This transformed problem is solvable by iteratively updating (ψ, θ) to decrease f while at the same time keeping the constraint $q \le 0$ satisfied by decreasing q whenever $q > 0$ in each step:

$$(\psi_{k+1}, \theta_{k+1}) \leftarrow (\psi_k, \theta_k) - \xi\omega_k \tag{7}$$

$$\text{where} \quad \omega_k = \arg\min_{\omega} ||\nabla f(\psi_k, \theta_k) - \omega||^2 \tag{8}$$

$$\text{s.t. } \langle \nabla q(\psi_k, \theta_k), \omega \rangle \ge \phi_k \tag{9}$$

this could be solved in closed form, which gives $\omega_k = \nabla f(\psi_k, \theta_k) + \lambda_k \nabla q(\psi_k, \theta_k)$, with $\lambda_k = \max\left(\frac{\phi_k - \langle \nabla f(\psi_k,\theta_k), \nabla q(\psi_k,\theta_k)\rangle}{\|\nabla q(\psi_k,\theta_k)\|^2}, 0\right)$ and ϕ_k is chosen to be $\eta q(\psi, \theta)$ or $\eta \|\nabla q(\psi, \theta)\|^2$ (we used $\eta \|\nabla q(\psi, \theta)\|^2$ in experiment and take $\eta = 0.5$). For the choice $\phi_k = \eta\|\nabla q(\psi_k, \theta_k)\|^2$, set $\lambda_k = 0$ when $\nabla q(\psi_k, \theta_k) = 0$. The constraint then reduces to $0 \ge 0$ and $\omega_k = \nabla f(\psi_k, \theta_k)$.

Therefore, the procedure to optimize f by jointly updating (ψ, θ) is:

$$(\psi_{k+1}, \theta_{k+1}) \leftarrow (\psi_k, \theta_k) - \xi[\nabla f(\psi_k, \theta_k) + \lambda_k \nabla q(\psi_k, \theta_k)] \tag{10}$$

where $\nabla f(\psi_k, \theta_k) = \nabla_{(\psi_k,\theta_k)} f(\psi_k, \theta_k)$ is the gradient update of outer problem and $\nabla q(\psi, \theta) = \nabla_{(\psi,\theta)} q(\psi, \theta)$ imposes the constraint. Expressed explicitly, the gradient of f and q are:

$$\nabla_{(\psi,\theta)} f(\psi_k, \theta_k) = \left(\lambda \nabla_\psi L(P_{\psi_k}, P), \nabla_\theta J(P, \pi_{\theta_k}) + \lambda \nabla_\theta L(P_{\psi_k}, P)\right), \tag{11}$$

$$\nabla_{(\psi,\theta)} q(\psi_k, \theta_k) \approx \begin{pmatrix} \nabla_\psi J(P_{\psi_k}, \pi_{\theta_k^W}) - \nabla_\psi J(P_{\psi_k}, \pi_{\theta_k}), \\ -\nabla_\theta J(P_{\psi_k}, \pi_{\theta_k}) \end{pmatrix}. \tag{12}$$

where θ_k^W is the W-step approximation of the optimal policy parameters $\theta^* \in \arg\max_{\theta'} J(P_\psi, \pi_{\theta'})$. Algorithm 1 uses value-based estimates of these directions, with the policy and normalization choices given in Appendix E.3.

D.3 Stochastic Transition Functions Case

A stochastic world model is fine-tuned by minimizing negative log likelihood,

$$\mathcal{L}(\psi; D) = -\frac{1}{|D|} \sum_{(s,a,s') \in D} \log P_\psi(s' \mid s, a). \tag{13}$$

For $s' \sim P(\cdot \mid s, a)$, its population counterpart differs from $D_{\mathrm{KL}}(P \| P_\psi)$ by an entropy term independent of ψ. The Stage 1 consistency loss is therefore

$$L(P_\psi, P) = \mathbb{E}_{(s,a)\sim P, \pi_{\theta(\psi)}} D_{\mathrm{KL}}(P(\cdot \mid s, a) \,\|\, P_\psi(\cdot \mid s, a)). \tag{14}$$

Let $\hat{P}_D(\cdot \mid s, a)$ denote an empirical conditional next-state distribution, where repeated samples or a conditional density estimate are available. Its prediction residual is $D_{\mathrm{KL}}(\hat{P}_D \| P_\psi)$. The Stage 2 gradients follow the same likelihood objective, with the data or target distribution in the first KL argument:

$$\begin{aligned} G_{\mathrm{real}} &= \nabla_\psi \frac{1}{|\tilde{D}|} \sum_{(s,a,\tilde{s}') \in \tilde{D}} -\log P_\psi(\tilde{s}' \mid s, a) \Big|_{\psi=\psi_0}, \\ G_{\mathrm{target}} &= \nabla_\psi \mathbb{E}_{(s,a)\sim D_{\mathrm{all}}} D_{\mathrm{KL}}\Big(P_{\hat{\psi}}(\cdot \mid s, a) \,\|\, P_\psi(\cdot \mid s, a)\Big)\Big|_{\psi=\psi_0}. \end{aligned} \tag{15}$$

A KL-based counterpart of the clean-residual penalty in Eq. 5 is

$$\begin{aligned} L_P^{\mathrm{sto}} = (1-\alpha)\big(1 - \cos(G_{\mathrm{real}}, G_{\mathrm{target}})\big) \\ + \frac{\alpha}{|D_p|} \sum_{(s_i,a_i)\in D_p} \Big[D_{\mathrm{KL}}\Big(\hat{P}_{\tilde{D}}(\cdot \mid s_i, a_i) \,\|\, P_{\psi_0}(\cdot \mid s_i, a_i)\Big) \\ - D_{\mathrm{KL}}\Big(\hat{P}_D(\cdot \mid s_i, a_i) \,\|\, P_{\psi_0}(\cdot \mid s_i, a_i)\Big)\Big]^2. \end{aligned} \tag{16}$$

Both residuals use the same pretrained reference. This penalty depends on the poisoned distribution and vanishes when it equals the clean distribution. The corresponding data-level deviation is

$$\delta_d = \frac{1}{|\tilde{D}|} \sum_{(s_i,a_i)\in\tilde{D}} D_{\mathrm{KL}}\Big(\hat{P}_{\tilde{D}}(\cdot \mid s_i, a_i) \,\|\, P_{\psi_0}(\cdot \mid s_i, a_i)\Big),$$

with $\delta_{d,\mathrm{ref}}$ obtained from D. For a T-transition rollout, the population model-level deviation is

$$\delta_m = \frac{1}{T} \sum_{t=0}^{T-1} D_{\mathrm{KL}}(P(\cdot \mid s_t, a_t) \,\|\, P_\psi(\cdot \mid s_t, a_t)).$$

Conditional-distribution KL measurements require finite divergences and estimable conditional distributions. The continuous latent-model experiments use the deterministic prediction-error metrics in Section 3.3.

E Implementation Details

We instantiate the experiment based on the TD-MPC2 framework. Thus all neural networks including policy, dynamics model, encoder and value function have MLP architecture with several layers, for detailed information about architecture, please refer to [Hansen et al., 2024].

E.1 Model Predictive Control

Algorithm 2 General Model Predictive Control (MPC)

Input: World model ψ, current state s_t, goal o_g (optional) horizon H, number of rollouts `num_samples`
Encode: $z_t = \text{enc}(s_t)$, $z_g = \text{enc}(o_g)$ if goal is given
for each MPC step **do**
    Sample `num_samples` candidate action sequences $\{a^i_{0:H-1}\}_{i=1}^{\texttt{num_samples}}$ (from a policy prior, an action sampler, or learned actor)
    for each sequence i **do**
        Roll out latent trajectory $\hat{z}^i_{1:H} = P_\psi(\hat{z}_t, a^i_{0:H-1})$
        Evaluate cost or return:

$$J^i = \begin{cases} \sum_{h=0}^{H-1} \gamma^h r(\hat{z}^i_h, a^i_h) + \gamma^H Q(\hat{z}^i_H) & \text{(value/bootstrap)} \\ \|\hat{z}^i_H - z_g\|^2 & \text{(goal-closeness)} \\ \text{other task-specific objective} \end{cases}$$

    Select best sequence(s) according to J^i
    Refit a sampling distribution to top-K sequences
return first action or first k actions from selected sequence or sampler

We provide a general formulation of Model Predictive Control (MPC), which abstracts across variants such as TD-MPC2 [Hansen et al., 2024], DINO-WM [Zhou et al., 2025]. In this section, we occasionally use $\hat{z}$ to denote an imagined latent, compared to the real latent z encoded from the real state s from the environment.

To model the dynamics, TD-MPC2 [Hansen et al., 2024] learns the world model $P_\psi(z'|z, a)$ that predicts next latent state given current latent state and action, where $z = enc(s)$ is the latent state from a learned encoder, and its agent plans by using the learned world model P_ψ together with a learned prior policy π_{prior}. At each planning step, the planner first obtains `num_pi_trajs` candidate trajectories by rolling out π_{prior} under transition P_ψ; these yield the policy-guided action sequences. The planner then samples an additional `num_samples` $-$ `num_pi_trajs` random action sequences from a stochastic proposal (e.g., i.i.d. or Gaussian-perturbed sequences). During the inner improvement/refit loop, the action components coming from the prior policy are treated as guidance and are typically held fixed, while only the remaining (random) action sequences are optimized. This hybrid design reduces search dimensionality and biases search toward plausible behavior while still allowing exploratory refinement.

DINO-WM [Zhou et al., 2025] does not use a learned prior. Instead, it draws many random action sequences, rolls each sequence forward in the world model P_ψ for horizon H, and scores each rollout by a final-state objective (the distance to a goal latent z_g, e.g., $J = \|\hat{z}_H - z_g\|_2$). The top-performing sequences are retained as elites and the sampling distribution is refit to those elites (CEM-style); this process repeats until the sampling distribution concentrates on sequences that reach the goal.

The above MPC algorithm highlights the shared structure: rolling out candidate action sequences over a planning horizon H using a learned world model P_ψ, evaluating them under a task-specific objective (e.g., bootstrapped return, goal closeness, or other criteria), and executing the first action of the best sequence. The number of sampled rollouts `num_samples` and the planning horizon H are hyperparameters that directly affect the quality of planning and computation cost. The general MPC procedure is summarized in Algorithm 2.

E.2 Hyperparameters

We report our SWAAP key hyperparameters in Table 2. Other parameters are the same as stated in TD-MPC2 [Hansen et al., 2024].

Table 2: TD-MPC2 settings for DMControl, MyoSuite and MetaWorld. The lower panel gives the Stage 1 iteration budget and consistency weight, the Stage 2 weight used for target selection, and the victim learning rate for each task.

Hyperparameter	Value	Description
Model Poisoning Hyperparameters		
`latent_dim`	512	Latent-state dimension.
H	3	MPC planning horizon.
`num_samples`	512	Candidate action sequences per planning step.
W	30	Inner surrogate-policy updates per outer iteration.
N	16	Minibatch contributions per outer update.
`sample_batch_size`	256	Horizon-length sequences per sampled minibatch.
`num_step`	500 / 100	Rollout length for DMControl / MyoSuite and MetaWorld.
Checkpoint interval	10	Outer iterations between saved target evaluations.
Direct evaluation	10	Episodes per saved Stage 1 checkpoint.
Data Poisoning Hyperparameters		
r_p	.1	Main poisoning fraction; the full grid uses .1, .2, .3.
α	.1, .5, .9	Sensitivity grid; Table 1 uses .9.
`poison_steps`	5,000	Projected gradient-descent updates.
`step_size`	10	Ordinary SWAAP construction step size.
ϵ	1	Clean-centered ℓ_∞ radius, followed by clipping to $[0, 1]$.
`noise_scale`	.3	Multiplicative random-target scale in Table 1.
`train_epochs`	10	Victim fine-tuning epochs.
`batch_size`	256	Victim minibatch size; the final partial batch is retained.
Optimizer updates	200	Victim updates per 5,000-row fit.
$\lvert D \rvert$	5,000	Adaptation-data size.
$\lvert D_{\text{all}} \rvert$	50,000	Reference-data size for G_{target}.

Task	Stage 1 iterations	λ	α (target selection)	Victim LR	Victim epochs
Humanoid-Walk	400	1	.9	10^{-4}	10
Humanoid-Run	400	1	.9	1.6×10^{-4}	10
Dog-Walk	800	1	.5	3.2×10^{-3}	10
Dog-Run	400	1	.9	10^{-4}	10
Cheetah-Run	800	0	.5	2×10^{-4}	10
Myo-Pen-Twirl-Hard	400	1	.5	8×10^{-4}	10
Myo-Reach-Hard	400	10	.5	6.4×10^{-3}	10
MW-Push	1,000	1	.9	3.2×10^{-3}	10
MW-Soccer	1,000	0	.9	10^{-4}	10
ManiSkill2 LiftCube	400	1	.5	1.75×10^{-4}	25
ManiSkill2 PickCube	400	1	.7	5×10^{-5}	50
ManiSkill2 StackCube	400	1	.7	2.5×10^{-4}	10

E.3 Stage 1: Target Model Identification

The model poisoning starts with a trained TD-MPC2 checkpoint (such as humanoid-walk-2.pt from the TD-MPC2 model base), from which the following components are extracted as input to Algorithm 1: the trained policy network π_θ as θ_0, the dynamics model P_ψ as ψ_0, the value network (a Q net in TD-MPC2), the reward model, and an encoder. Stage 1 updates copies of the dynamics and surrogate policy; the Original dynamics and policy, encoder, reward, termination and value-network weights remain fixed. The model-based agent consists of a prior policy θ and transition model ψ, and $\pi_{\theta(\psi)}$ stands for the agent's policy after MPC planning with θ and ψ.

During each iteration, imagined rollouts from $(P_{\psi_k}, \pi_{\theta_k})$ form $\mathcal{B}_{kk}$. We copy θ_k to θ_k^0 and perform $W = 30$ clipped Adam ascent steps, retaining the inner optimizer's moments between outer iterations. Rollouts from $(P_{\psi_k}, \pi_{\theta_k^W})$ then form $\mathcal{B}_{kW}$. Real-environment rollouts using prior policy θ_k and planning dynamics ψ_0 form $\mathcal{B}_{pk}$; separate clean rollouts using (ψ_0, θ_0) supply the consistency loss.

Given data (s, a, s'), the policy-update direction differentiates the TD-MPC2 `update_pi` objective: the value estimate $Q(s, a_{pred})$ and scaled action entropy, where a_{pred} is sampled from the policy being

updated. The dynamics-update direction differentiates $Q(s'_{pred}, a'_{pred})$ through $s'_{pred} = P_\psi(s, a)$ and the next action sampled from θ_k. Thus θ_k^W generates $\mathcal{B}_{kW}$, while θ_k supplies the next-state actions in both dynamics-update directions. Sampled sequences use temporal weights 0.5^t and entropy coefficient 10^{-4}. Each helper updates the shared Q scale from the first time step's 95th–5th percentile span, lower-bounded by 1, with moving-average coefficient .01. Each division uses a detached clone of that call's scale. The consistency loss averages squared prediction errors over clean samples and latent coordinates. The pretrained Q network remains fixed throughout these updates. The improved-policy samples change the states used by the dynamics update, while its next-state actions still come from θ_k. These fixed-critic objectives and the per-call normalization are the approximations used in Algorithm 1.

Each outer update averages $N = 16$ separately sampled minibatch directions. The outer direction combines $\lambda \nabla_\psi L$ with the policy gradient from $\mathcal{B}_{pk}$; the constraint direction combines the difference of the two transition-gradient estimates with the negative policy gradient from $\mathcal{B}_{kk}$. For each minibatch, the barrier multiplier uses the squared joint norm $\|\nabla_\psi q\|_2^2 + \|\nabla_\theta q\|_2^2$ in both numerator and denominator, with $\eta = .5$, a denominator stabilizer 10^{-8}, and a lower clamp at zero. The multiplier is detached. We average the resulting directions and apply separate clipped Adam descent steps to ψ and θ. All three Stage 1 optimizers use learning rate 3×10^{-4} and gradient-norm clipping at 20.

During Stage 1, we use the iteration counts in Table 2 and evaluate $J(P, \pi_{\theta_0}(\psi_k))$, the return of the pretrained policy using the perturbed model ψ_k for planning in the real environment, for ten episodes every ten iterations. These measurements characterize the return–δ_m trade-off, as illustrated in Figure 5. The selected dynamics model supplies the target predictions for Stage 2; target selection follows Appendix E.4.

E.4 Stage 2: Data Poisoning

We collect a reference data D_{all} of 50,000 transitions and an adaptation data D of 5,000 transitions. Data collection uses the pretrained agent, replacing each action with a uniformly random action with probability .03. A trajectory contains 500 decisions for DMControl and 100 for MyoSuite and MetaWorld.

Poison construction. The support comprises the $k = \lfloor r_p |D| \rfloor$ transitions with the largest Original–target prediction disagreements. Both matching losses average over rows and latent coordinates; the regularizer in Eq. 5 averages the squared differences of residual magnitudes over the selected rows. The support is fixed throughout construction. We initialize $\tilde{s}'_i = s'_i$ and take 5,000 gradient-descent steps of size $\eta = 10$. For $i \in S$, each step applies

$$
\begin{aligned}
u_i^{(j)} &= \tilde{s}_i'^{(j)} - \eta \nabla_{\tilde{s}'_i} L_P, \\
\tilde{s}_i'^{(j+1)} &= \mathrm{clip}_{[0,1]^d}\left[s'_i + \mathrm{clip}_{[-\epsilon,\epsilon]^d}\left(u_i^{(j)} - s'_i\right) \right], \qquad \epsilon = 1.
\end{aligned}
$$

The inner clipping operation bounds the coordinatewise change from the clean label, and the outer clipping enforces the latent coordinate range. For clean labels in $[0, 1]^d$, the chosen $\epsilon = 1$ adds no restriction beyond that range. The projection does not enforce the encoder's group-normalization constraints. Unselected labels, current-state inputs, actions, rewards, termination fields, and row order retain their clean values.

Victim fine-tuning. Each fit initializes dynamics from the task's pretrained checkpoint and uses fresh Adam state at the clean-calibrated learning rate in Table 2. The encoder, reward model, actor, and critics remain fixed. Clean and poisoned fits share ten saved row permutations, one per epoch. Each epoch processes 19 batches of 256 rows and one batch of 136 rows, giving 200 updates. Every final fine-tuned model is evaluated on 100 paired environment and planner random seeds. Return sums rewards over 500 DMControl decisions or 100 MyoSuite/MetaWorld decisions.

Target selection. A Stage 1 checkpoint is eligible when its ten-episode Direct return is at most .9 times the same run's positive Original return and its rollout δ_m is at most .30. Runs with at least five distinct eligible checkpoints supply at most 20 candidate targets. Each candidate receives one poison construction and one common development fine-tuning run. The three candidates with the largest matched Clean-minus-SWAAP return loss receive two further common development fine-tuning runs. We select the target with the largest mean loss over the three runs, resolving ties by the original

candidate ranking and checkpoint identity. Three separate final fine-tuning runs, evaluated on episodes disjoint from development, measure the selected poison. We retain three accepted Stage 1 runs per task for DMControl, MyoSuite and MetaWorld, with the pretrained checkpoint and data fixed within each task.

For ManiSkill2, one retained target per task was selected using one development fine-tuning run. The other five contributing targets were selected using three development fine-tuning runs. All eight selected poisons receive three separate final fine-tuning runs. These results are conditional on the retained selections, with two contributing Stage 1 runs for Lift and three each for Pick and Stack.

Aggregation and operating points. Return and δ_m are first averaged over the 100 episodes of each fine-tuned model, then over the three fine-tuning runs for each Stage 1 run. Task summaries give the mean and sample SD of these Stage 1-run means. These levels distinguish target-search repetitions, fine-tuning randomness and variation across test episodes. Additional Stage 1 runs provide more candidates for an attacker with a larger search budget; the reported averages retain all contributing runs. δ_d is measured once per poisoned dataset over all 5,000 transitions; $\delta_{d,\mathrm{ref}}$ uses the fixed clean data. The implementation adds 10^{-8} to the normalization denominators of δ_d and δ_m. For the uniform-$\alpha = .9$ results, we compute each fine-tuning run's percentage reduction from its paired Clean and poisoned mean returns, then average over fine-tuning runs and Stage 1 runs. Target selection used $\alpha = .5$ for Myo-Pen-Twirl-Hard, Myo-Reach-Hard, Dog-Walk, and Cheetah-Run, and $\alpha = .9$ for the other five tasks. Table 1 presents the .9 operating point chosen after the sensitivity grid, reusing the selected targets, supports, and final evaluation panels.

SWAAP (Random). The random baseline uses multiplicative target perturbations $s' \odot (1 + \epsilon)$ with coordinates of ϵ sampled uniformly from $[-.3, .3]$. It uses construction step size 10 and Bernoulli row selection with probability r_p. The main comparison uses $\alpha = .9$. The randomly selected transitions are independent of SWAAP's selected set. Fixed-size and shared-transition comparisons are reported in Appendix F.3.

E.5 Hardware Specification and Computational Cost

We use NVIDIA A100, GeForce RTX 3090, and GeForce RTX 5080 GPUs. The local RTX 5080 workstation has 16 GB GPU memory, an Intel Core Ultra 9 285K CPU, and 64 GB RAM. Each reported GPU-hour denotes one hour of allocated time on one GPU, including startup, collection, optimization, and evaluation within that allocation.

Per-stage workload and runtime. A Stage 1 search uses the task-specific iteration budget in Table 2. One recorded Humanoid-Walk search with 400 outer iterations took 101.08 minutes, including data collection, optimization, checkpointing, and 410 evaluation episodes across the Original baseline and 40 saved targets. Subsequent candidate screening is separate.

Each ordinary Stage 2 construction performs 5,000 projected updates on 500 selected successor labels in a 5,000-transition data. A single Stage 2 Poison construction takes around 15 minutes.

For the same Humanoid-Walk pool, on an A100 80 GB PCIe GPU, a victim fit with ten epochs, batch size 256, and 200 optimizer updates took a median of 1.31 seconds across three final fitting orders (range: 1.29–1.45 seconds). This timer includes optimization and checkpoint writing, but excludes model/data loading and optimizer initialization. Evaluating each fitted endpoint on 100 episodes of 500 control decisions took a median of 315.13 seconds (5.25 minutes; range: 314.66–316.41 seconds), obtained by summing the recorded episode timers. These evaluation timings include planning and environment stepping but exclude model loading, environment creation/reset, additional reference scoring, and output writing. These are representative task-specific measurements with distinct timer boundaries, not universal stage latencies or an end-to-end attack-time estimate.

Repeated-experiment accounting. For the nine DMControl, MyoSuite, and MetaWorld tasks, target search and the primary SWAAP evaluations used 243.25 GPU-hours. Additional comparisons and saved-trajectory metrics used 25.34 GPU-hours, and TRIM used 6.98 GPU-hours, totaling 275.57 GPU-hours for these experimental blocks. The α–r_p sweep used a further 169.63 GPU-hours for 471 new conditions, with three fine-tuning runs and 100 test episodes per condition and run; 27 existing conditions were reused. The main-experiment total includes recorded failed allocations, and reused conditions are not charged again. These allocation totals cover the stated repeated experiments,

including candidate screening and confirmation, rather than the runtime of one attack or the complete computational cost of every study in the paper.

F Main Extended Results

F.1 SWAAP and SWAAP (Random)

Figure 4 compares SWAAP and SWAAP (Random) over $\alpha \in \{.1, .5, .9\}$ and $r_p \in \{.1, .2, .3\}$. SWAAP reuses its selected Stage 1 targets throughout the sweep. Support size varies with r_p, and poisoned labels are optimized for each condition. Larger α generally reduces poisoned-data deviation, while its effect on return depends on the task and target.

Increasing r_p has task-dependent effects. At $\alpha = .1$, mean Humanoid-Walk return decreases from 333.1 to 289.7 and 255.2 as r_p increases from .1 to .2 and .3. MW-Soccer instead increases from 890.5 to 1393.7 and 1480.6. These comparisons use targets selected at $r_p = .1$ and a fixed 5,000-step construction budget. They characterize the sensitivity of the implemented attack to support size and regularization.

F.2 Repeated Fine-Tuning Experiments

We repeat fine-tuning for each poisoned dataset used in Table 1. Each run starts from a fresh copy of the same pretrained world model and a new optimizer, with a different random permutation of the training data. Clean and poisoned conditions use matching permutations and evaluation random seeds.

Table 3 separates the results by Stage 1 run. Each entry averages three fine-tuning runs, each evaluated for 100 episodes. Table 1 averages these Stage 1-run means and reports their sample standard deviation. Additional Stage 1 restarts give the attacker more candidate targets to compare using development episodes. The final test episodes measure performance after the poison is fixed, with evaluation random seeds chosen independently of target selection.

Table 3: Repeated fine-tuning of the SWAAP datasets in Table 1, with the same targets and task-specific settings. Entries are cumulative test Return, reported as mean $\pm$ sample SD across three fine-tuning runs of each fixed poisoned dataset; each model is evaluated for 100 episodes. The three columns correspond to independent Stage 1 runs.

Task	Stage 1 run 1	Stage 1 run 2	Stage 1 run 3
Humanoid-Walk	749.1 ± 7.3	316.5 ± 25.0	95.5 ± 18.2
Humanoid-Run	367.9 ± 45.8	416.0 ± 20.9	263.2 ± 32.8
Dog-Walk	781.1 ± 15.9	728.2 ± 22.0	665.6 ± 35.4
Dog-Run	364.0 ± 8.7	572.5 ± 4.2	585.9 ± 11.7
Cheetah-Run	739.8 ± 8.5	820.3 ± 5.6	802.0 ± 10.5
Myo-Pen-Twirl-Hard	3333.7 ± 229.4	3369.9 ± 69.2	3150.3 ± 215.5
Myo-Reach-Hard	662.7 ± 20.5	614.9 ± 37.5	758.4 ± 5.9
MW-Push	1135.0 ± 153.0	1684.3 ± 28.6	1408.6 ± 48.8
MW-Soccer	900.4 ± 83.2	1050.6 ± 69.1	1489.6 ± 50.7
Lift-Cube	191.3 ± 0.1		192.1 ± 0.3
Pick-Cube	156.7 ± 3.1	159.2 ± 3.6	160.5 ± 0.4
Stack-Cube	160.2 ± 2.5	165.8 ± 2.9	141.6 ± 2.2

F.3 Transition Selection in SWAAP (Random)

SWAAP (Random) in Table 1 chooses the transitions to perturb randomly, whereas SWAAP selects them using the Stage 1 target model. We examine whether this difference affects the comparison. We construct two SWAAP (Random) datasets: one changes 500 randomly selected transitions, and the other changes exactly the same 500 transitions selected by SWAAP. Each transition has the same randomly perturbed next-state target in both versions, so only the choice of transitions changes.

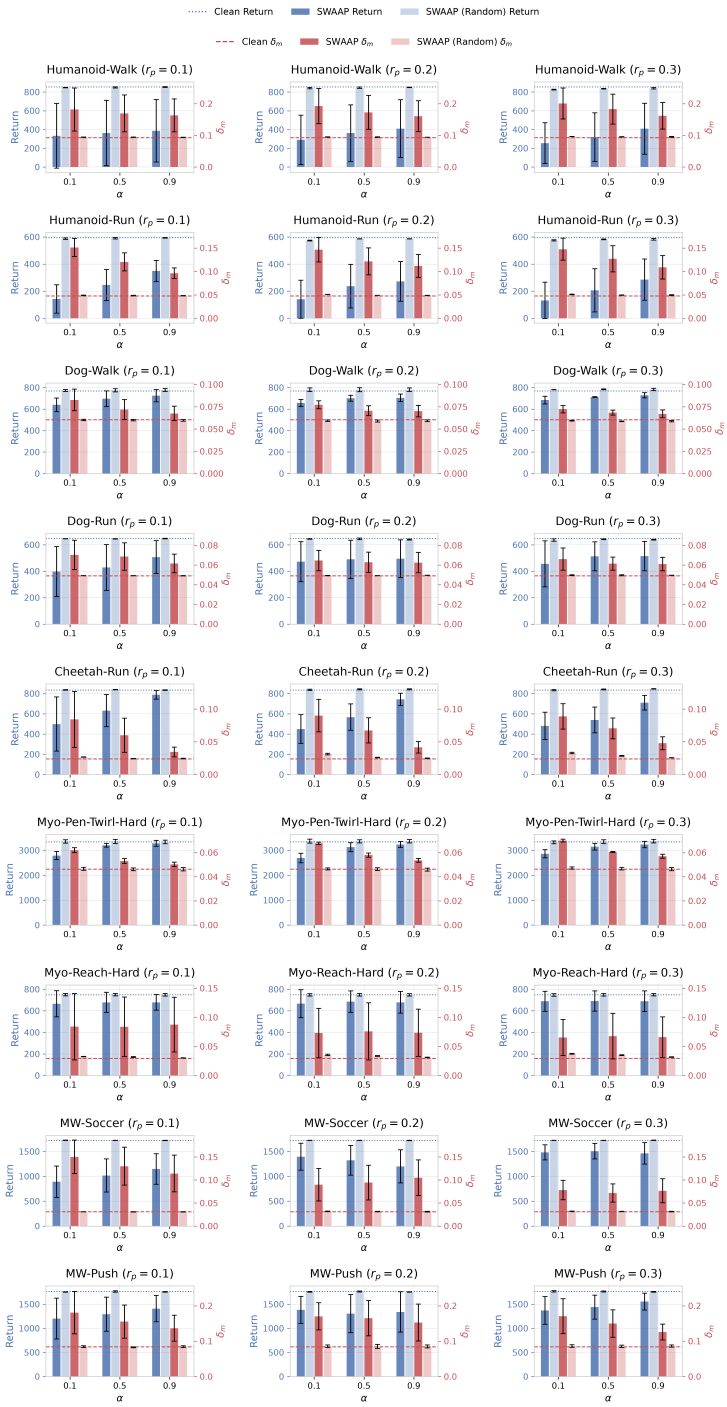


Figure 4: SWAAP and SWAAP (Random) across poisoning fractions r_p (columns) and stealth weights α (horizontal axes). Blue bars show cumulative Return and red bars show rollout prediction error δ_m; darker and lighter bars denote SWAAP and SWAAP (Random). The dotted blue line is matched Clean Return; the dashed red line is the clean-fitted model's prediction error on its own rollouts. Bars and capped error bars are means and sample SDs across three Stage 1 runs, each averaging three fine-tuning runs with 100 episodes per model. SWAAP targets are fixed across the grid, and each task uses common axes across its three r_p panels.

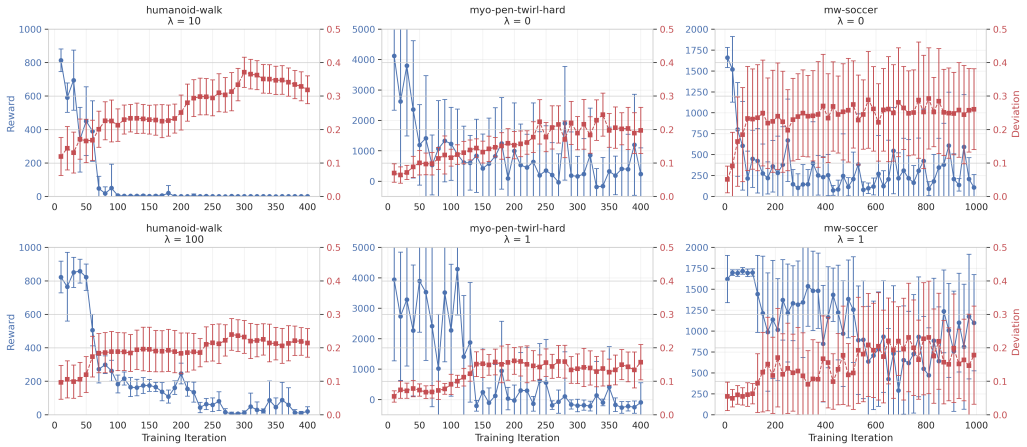


Figure 5: Model poisoning curves (referred to as Direct Model Poisoning in Table 1) of return and δ_m tested across different training iterations under different λ, each testing is the average of 10 test episodes, and thus each data point is a single run with 10 episodes. From these plots, we observe that λ is suppressing the deviation of the model, but the varying influence of λ over different environments implies that a different λ is required to constrain the perturbation of the model.

Table 4 reports the results with $r_p = .1$ and $\alpha = .9$. The two step-size-1 columns compare transition selection. We then use the same selected transitions and construction step size 10 to compare random and learned targets. The main-table random baseline instead uses Bernoulli selection and step size 10.

Table 4: Cumulative test Return under different transition selections and construction step sizes, with $r_p = .1$ and $\alpha = .9$. The three SWAAP (Random) columns use the same random target field; all methods modify exactly 500 transitions. Values average 100 test episodes per fine-tuned model, three fine-tuning runs per target and three Stage 1 runs. Lower Return means greater degradation.

	SWAAP (Random)			SWAAP
Task	Random transitions step size 1	SWAAP transitions step size 1	SWAAP transitions step size 10	SWAAP transitions step size 10
Humanoid-Walk	856.5	836.9	818.6	387.0
Dog-Run	644.2	639.3	636.8	507.5
MW-Push	1764.6	1763.3	1759.4	1409.3

Using SWAAP's selected transitions lowers the random-target return, but the learned targets cause substantially greater degradation. This difference remains when the construction step size is matched: SWAAP has lower return in all nine paired Stage 1 comparisons.

F.4 Model Poisoning Training Curve

We show the stage one model training curve in Figure 5, which comes from testing the agent that directly uses the perturbed model for 10 episodes after every few iterations of updates. From these plots, we observe that λ suppresses the model's deviation, but the varying influence of λ across different environments implies that different λ values are required to constrain the model's perturbation.

F.5 Pre-Training Detection

Figure 6 compares mean data-level deviation before and after poisoning across values of α. Larger α generally brings the poisoned-data residual closer to the clean reference. Residual-CUSUM separately evaluates whether these changes produce detectable trajectory-level alarms.

Table 5: Poisoned dataset residual-CUSUM detection with task-specific clean-policy calibration. The nine-task block averages three poisons per task and α, one from each independent Stage 1 run. The ManiSkill2 block averages the final selected poisons: two for Lift and three each for Pick and Stack. One of Lift's three accepted runs produced no stealth-qualified candidate. ManiSkill2 selection uses these same δ_d and TPR_5 measurements, so its reported TPR_5 reflects the selection criterion. Each poison is scored once before fitting, using all ten DMControl or 50 MyoSuite/MetaWorld/ManiSkill2 trajectories. AUC is full ROC-AUC. pAUC_5/pAUC_{10} are McClish-standardized partial AUCs over FPR $\leq .05/.10$ (chance $= 0.5$). TPR_5/TPR_{10} use fixed nominal $5\%/10\%$ thresholds. FPR_5/FPR_{10} are measured on a separate 1,000-trajectory clean test set per task, reused across poisons.

	$\alpha = 0.1$			$\alpha = 0.9$			Clean FPR	
Task	AUC	$\mathrm{pAUC}_5/_{10}$	$\mathrm{TPR}_5/_{10}$	AUC	$\mathrm{pAUC}_5/_{10}$	$\mathrm{TPR}_5/_{10}$	FPR_5	FPR_{10}
MW-Push	.941	.684/.776	.72/.84	.538	.502/.512	.08/.14	.052	.109
Humanoid-Walk	.772	.585/.612	.27/.40	.526	.487/.480	.00/.10	.028	.100
Humanoid-Run	.910	.814/.842	.77/.77	.593	.562/.576	.20/.30	.055	.091
Dog-Walk	.845	.523/.586	.20/.37	.513	.487/.474	.00/.00	.050	.080
Dog-Run	.916	.589/.638	.27/.43	.468	.526/.520	.10/.10	.036	.072
Cheetah-Run	.715	.505/.534	.13/.23	.557	.487/.474	.00/.10	.056	.126
Myo-Pen-Twirl-Hard	.698	.650/.681	.42/.51	.478	.505/.506	.04/.12	.047	.087
Myo-Reach-Hard	.593	.500/.491	.04/.04	.407	.494/.488	.02/.04	.031	.079
MW-Soccer	.910	.510/.608	.15/.55	.474	.492/.483	.02/.04	.052	.086

ManiSkill2: Task (α)	AUC	$\mathrm{pAUC}_5/_{10}$	$\mathrm{TPR}_5/_{10}$	FPR_5	FPR_{10}
Lift-Cube (.5)	.657	.505/.503	.08/.14	.075	.124
Pick-Cube (.7)	.712	.504/.519	.07/.17	.044	.105
Stack-Cube (.7)	.572	.495/.496	.06/.12	.067	.115

F.5.1 Residual-Based CUSUM Pre-Training Detector

We evaluate a CUSUM-style sequential detector to screen candidate fine-tuning trajectories before they are used for model updates. Instead of using the latent-state norm $\|z_{t+1}\|_2$, we use a transition-residual statistic measured against the clean reference world model:

$$x_t = \|z_{t+1} - P_{\psi_0}(z_t, a_t)\|_2.$$

This statistic is aligned with pre-training detection because a defender with a reference world model can test whether each observed transition is anomalous relative to the predicted next latent state.

Given clean calibration trajectories, we estimate a robust center and scale using the median m and median absolute deviation MAD of the residual signal. For each candidate trajectory, we form centered residuals $r_t = x_t - m$ and update a one-sided CUSUM statistic

$$S_0 = 0, \qquad S_t = \max(0, S_{t-1} + r_t - \kappa), \qquad \kappa = 6\,\mathrm{MAD}.$$

The MAD is lower-bounded by 10^{-8}. The trajectory-level detection score is $\max_t S_t$.

The fitting, threshold-calibration, and clean-evaluation trajectories are mutually disjoint. Depending on available clean data, (300)–(500) trajectories are used to estimate the residual center and scale, followed by (500) new clean trajectories for task-specific threshold calibration and (1,000) independently seeded clean trajectories for final false-positive evaluation. Two thresholds are calibrated separately for nominal (5%) and (10%) clean FPR and are frozen before any poisoned trajectories are evaluated.

We report full ROC-AUC, McClish-standardized partial AUC over FPR ranges ([0,.05]) and ([0,.10]), TPR at the corresponding fixed thresholds, and the realized FPR on the independent clean test set. Standardized pAUC has chance value (0.5). The realized clean FPR need not equal its nominal calibration target exactly because it is measured on a separate finite test set. For the DeepMind Control poison data, only ten poisoned trajectories are available, so thresholded TPR changes in increments of 0.1 and has wide uncertainty. We report AUC and low-FPR pAUC alongside these coarse alarm fractions.

Table 5 shows that residual-CUSUM detectability depends on the task and stealth weight. At $\alpha = .1$, MW-Push has alarm rates of $72\%/84\%$ at the nominal $5\%/10\%$ thresholds, while Dog-Walk has

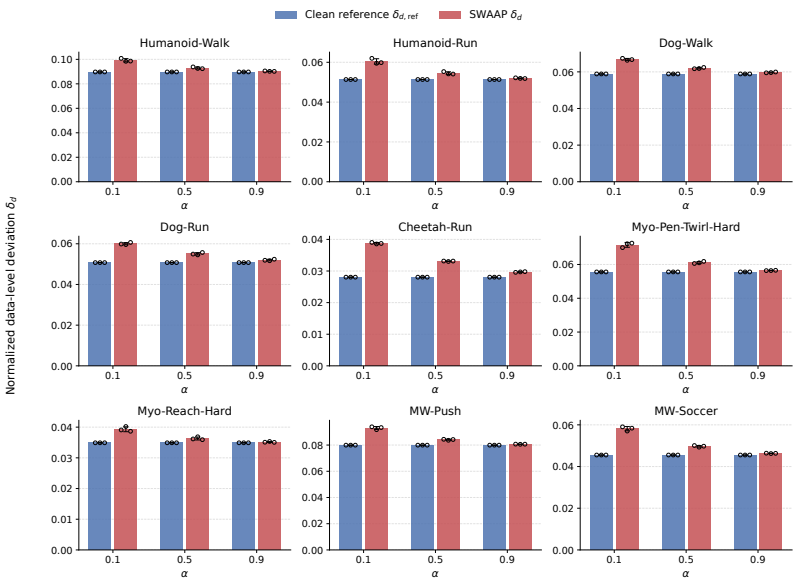


Figure 6: Pre-training data-level deviation across stealth weights α. Blue bars show the clean reference $\delta_{d,\mathrm{ref}}$ and red bars show poisoned-data δ_d. Bars are means across three Stage 1 datasets, caps show their sample SDs, and hollow points show the three dataset means. Each dataset mean uses all 5,000 transitions.

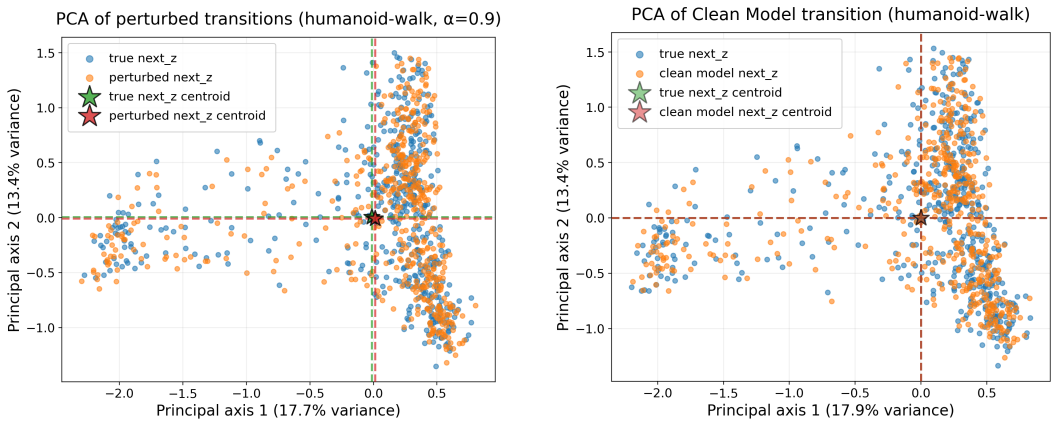
$20\%/36.7\%$ and MW-Soccer has $15.3\%/54.7\%$. Increasing α to .9 reduces score separation, with a mean AUC of .506 across the nine tasks. At the nominal 5% threshold, each poisoned data has the same alarm rate as its corresponding unmodified data. Humanoid-Run's 20% rate, for example, is already present in two of its ten clean source trajectories. The independent clean-test false-positive rates are reported separately in the table.

For ManiSkill2, the detector uses 500 clean trajectories for residual statistics, 500 for thresholds and 1,000 for independent false-positive evaluation. Each poisoned dataset contains 50 trajectories. At the settings in Table 1, mean alarm rates at the nominal 5% threshold are 8.0%, 6.7% and 6.0% on Lift, Pick and Stack, respectively, compared with independent Clean rates of 7.5%, 4.4% and 6.7%. These means use the same contributing Stage 1 runs as Table 1. Candidates were selected using this detector's 5% operating point, so these rates characterize the selected attacks rather than an independent stealth test. Lift and Pick have full AUCs of .657 and .712, respectively, showing separation outside that operating point.

F.5.2 Latent-Space PCA Visualization

We additionally visualize data-level stealthiness using PCA in the latent space used by the world model. This visualization is intended as a qualitative sanity check complementary to the scalar deviations in Table 1. For Humanoid-Walk with $\alpha = 0.9$, we take the 500 transitions selected for poisoning and compare their clean next-state latents z', poisoned next-state latents $\tilde{z}'$, and clean/reference-model predicted next-state latents $P_{\psi_0}(z, a)$. The poisoned-target plot illustrates the deviation from true next-states, while the clean-model-prediction plot illustrates clean prediction residuals used in $\delta_{d,\mathrm{ref}}$.

To make the visualization comparable, PCA is fitted on the pooled latent samples used in each panel and the first two principal components are plotted. Figure 7 shows that poisoned next-state targets largely overlap with their clean next-state counterparts and that their centroid remains close to the clean centroid. The clean/reference-model predictions show a similar overlapping pattern relative to true next states. This supports the interpretation that SWAAP's poisoned targets are not obvious latent outliers under this two-dimensional projection. This two-dimensional projection provides a qualitative view of the perturbations. Detectability is evaluated separately using the prediction residuals and CUSUM.



(a) Poisoned targets vs. clean next states

(b) Reference-model predictions vs. clean next states

Figure 7: Latent-space PCA visualization on Humanoid-Walk. (a) compares poisoned next-state targets $\tilde{z}'$ with their clean sources z', illustrating the data-level perturbation relative true states, note this is different from δ_d. (b) compares pretrained-model predictions $P_{\psi_0}(z, a)$ with true next states z', illustrating the clean prediction residuals used in $\delta_{d,\mathrm{ref}}$. Poisoned targets largely overlap with clean next states, similar to the clean model's natural prediction residuals.

F.6 Stage 1–Stage 2 Realization Bottleneck

SWAAP separates target identification from target realization. Stage 1 searches for a target world model $P_{\hat{\psi}}$ that induces low return while remaining close to clean dynamics. Direct Model Poisoning evaluates this target model directly. Stage 2 attempts to induce its effect through a limited set of poisoned transitions.

Two mechanisms can contribute to task-dependent effects. First, prediction errors affect control when they change decisions made by the policy or planner. Harmful changes may therefore occur at different states and deviation levels. Second, a harmful target may be difficult to learn from the permitted transitions. The unchanged data contribute to the victim gradient. Cosine matching aligns gradient directions at ψ_0 while leaving their magnitudes unconstrained, and those directions can change during fine-tuning. These mechanisms can limit Stage 2 even when the target itself is harmful.

Figure 3 illustrates the latter gap for the displayed targets. SWAAP approaches some Humanoid-Walk Direct returns, while Myo-Pen-Twirl-Hard has larger gaps. These selected examples show variation in realization. Separating these mechanisms across tasks requires matched model-deviation budgets and data controls.

We define **DMP-data** by replacing the selected top-r_p successors with predictions from $P_{\hat{\psi}}$, without gradient matching. Table 1 reports this data's deviation as δ_d^*, while its Direct Return remains the target-model evaluation. Using the same selected targets and three fine-tuning runs per Stage 1 run, uncontracted DMP-data has lower mean Return than the Table 1 SWAAP operating points on six of the nine DMControl, MyoSuite and MetaWorld tasks. For example, DMP-data gives 416.5, 351.1 and 1234.2 on Humanoid-Walk, Dog-Run and MW-Push, compared with 387.0, 507.5 and 1409.3 for SWAAP at $\alpha = .9$. Their corresponding DMP-data/SWAAP deviations are $.09008/.09030$, $.05476/.05199$ and $.08609/.08065$. This comparison has task-dependent differences in both efficacy and data deviation.

We also compare the methods at matched mean data deviation using a separate selected target per task. DMP-data labels are interpolated toward their clean values, with the coefficient chosen before fine-tuning so that their mean δ_d does not exceed SWAAP's. Table 6 reports all nine tasks. SWAAP causes greater mean return reduction on MW-Push, MW-Soccer, Myo-Reach-Hard and Humanoid-Run, while DMP-data is stronger on the other five tasks. This comparison controls mean data error rather than rollout error or detector alarms.

Table 6: DMP-data and SWAAP at matched mean data deviation. Reduction is the mean percentage decrease in cumulative Return relative to each method's matched Clean fit over three fine-tuning runs of one fixed target per task. Larger reduction means greater degradation. SWAAP uses $\alpha = .5$ for MyoSuite, Dog-Walk and Cheetah-Run, and .9 elsewhere. DMP-data is contracted toward Clean to satisfy $\delta_d^{\mathrm{DMP}} \leq \delta_d^{\mathrm{SWAAP}}$. Cheetah-Run already satisfies the inequality without contraction.

	δ_d		Reduction (%)	
Task	SWAAP	DMP-data	SWAAP	DMP-data
MW-Push	0.080459	0.080459	27.49	7.31
MW-Soccer	0.046158	0.046158	11.17	2.74
Myo-Pen-Twirl-Hard	0.060227	0.060227	0.69	2.00
Myo-Reach-Hard	0.037168	0.037168	1.35	0.08
Humanoid-Walk	0.090504	0.090504	77.84	90.20
Humanoid-Run	0.052045	0.052045	24.44	20.43
Dog-Walk	0.062007	0.062007	5.31	8.70
Cheetah-Run	0.032298	0.030934	3.51	5.62
Dog-Run	0.052280	0.052280	5.17	10.40

Table 7: TRIM at $r_p = .1$ and trimming fraction $\beta = .2$. Entries are cumulative test Return, reported as mean $\pm$ sample SD across three Stage 1 runs after averaging three fine-tuning runs per selected target and 100 episodes per model. SWAAP uses $\alpha = .9$ for Humanoid-Walk and Dog-Run and $\alpha = .5$ for Myo-Reach-Hard. Clean uses unmodified data. Both procedures use ten epochs: TRIM retains 4,000 transitions per round for 160 updates; fitting without TRIM uses 5,000 transitions for 200 updates.

Task	Stage 1 seeds	Clean, no TRIM	Clean, TRIM	SWAAP, no TRIM	SWAAP, TRIM
Humanoid-Walk	3	856.9 ± 3.6	857.6 ± 2.3	387.0 ± 332.5	483.8 ± 305.2
Dog-Run	3	647.9 ± 0.1	646.5 ± 1.3	507.5 ± 124.4	613.3 ± 31.8
Myo-Reach-Hard	3	749.2 ± 10.6	618.7 ± 7.8	678.8 ± 93.0	633.4 ± 35.9

These selected targets and settings differ from the three-Stage 1-run comparison above. On the four $\alpha = .5$ tasks, the methods were evaluated on different GPU types with separate matched Clean controls.

DINO-WM further illustrates limited data realization. Direct predictor replacement changes test success from $16/20$ to $1/20$, while SWAAP produces smaller losses (Appendix H.1). In a separate development comparison on the same 320-transition bank, exact target labels on all transitions give $1/20$ and $0/20$ successes against Clean $13/20$ and $18/20$. Copying labels onto only the global top 32 gives $14/20$ and $17/20$. Full-bank copying demonstrates a learnable harmful target, but changes both the edited fraction and perturbation magnitude. It is a diagnostic rather than an upper bound on SWAAP efficacy.

F.7 TRIM Defense results

We evaluate TRIM with $\beta = .2$ on Humanoid-Walk, Dog-Run, and Myo-Reach-Hard, using three independent Stage 1 runs and three fine-tuning runs per selected target. Clean and poisoned data receive the same defense. Both fitting procedures use ten epochs: TRIM retains 4,000 rows per round for 160 updates, while ordinary fitting uses all 5,000 rows for 200 updates. This comparison evaluates TRIM at equal epochs. Its effect combines transition removal with fewer optimizer updates, rather than isolating removal at an equal update count.

TRIM increases poisoned return by 96.7 ± 27.3 points on Humanoid-Walk and 105.8 ± 136.3 on Dog-Run. The corresponding Clean-return changes are $+0.6 \pm 5.1$ and -1.4 ± 1.1 points. Humanoid-Walk retains a defended Clean-minus-SWAAP gap of 373.8 ± 303.9 points. Dog-Run's gap is 33.2 ± 32.8. On Myo-Reach-Hard, poisoned return decreases by 45.4 ± 112.1 points and Clean return decreases by 130.5 ± 5.3 points. The small signed gap between its two defended conditions accompanies substantial benign utility loss. The paired-effect SDs summarize variation across Stage 1-run means.

Table 8: Offline iterative-FGSM/ℓ_∞-PGD on MW-Push at $r_p = .1$, compared with SWAAP at $\alpha = .9$. PGD modifies next-state targets to reduce the value of a clean prior policy. Return is cumulative test return, δ_m is rollout prediction error, and δ_d is data prediction error. Return and δ_m are mean $\pm$ sample SD across the three Stage 1 comparison runs, each averaging three fine-tuning runs with 100 episodes per model. δ_d averages three poison data; Clean uses the unmodified data.

Method	Return	δ_m	δ_d
Clean	1764 ± 12	0.084 ± 0.004	0.0799
PGD ($\epsilon = 0.015$)	1776 ± 2	0.084 ± 0.005	0.0829
PGD ($\epsilon = 0.05$)	1736 ± 18	0.099 ± 0.005	0.0984
SWAAP	1409 ± 275	0.137 ± 0.036	0.0807

F.8 Iterative FGSM/PGD Transition-Perturbation Baselines

We compare SWAAP with iterative signed-gradient transition-perturbation baselines adapted from adversarial state-perturbation attacks. These baselines do not identify a target world model or match fine-tuning gradients. Instead, they directly perturb next-state targets to lower the value of a prior policy π_θ trained from the clean world model.

For a clean transition (s, a, s'), we initialize $\tilde{s}'_0 = s'$ and run K projected signed-gradient steps:

$$\tilde{s}'_{k+1} = \Pi_{[s'-\epsilon, s'+\epsilon]} \left(\tilde{s}'_k - \eta \, \mathrm{sign} \left(\nabla_{\tilde{s}'_k} Q(\tilde{s}'_k, \pi_\theta(\tilde{s}'_k)) \right) \right), \qquad k = 0, \ldots, K-1,$$

where η is the step size, ϵ bounds the maximum per-coordinate perturbation, and $\Pi_{[s'-\epsilon, s'+\epsilon]}$ denotes projection to the ℓ_∞ box around the original next state. When $K = 1$, this reduces to an FGSM-style perturbation; for $K > 1$, it is an iterative-FGSM / ℓ_∞ PGD-style baseline.

We evaluate two variants. The **offline transition-perturbation baseline** applies this replacement to fine-tuning transitions before model update, matching our data-poisoning setting. The **online state-perturbation baseline** applies an analogous perturbation at test time to the state observed by the victim. The online variant is not the same threat model as SWAAP because it requires test-time intervention, but it provides context against a standard adversarial-state attack.

Table 8 shows that the offline signed-gradient transition-perturbation baseline is weaker than SWAAP on MW-Push. Increasing ϵ from .015 to .05 lowers its return from 1776 to 1736 and increases δ_d from .0829 to .0984. In contrast, SWAAP achieves a return of 1409 with $\delta_d = .0807$, close to the clean reference of .0799. SWAAP uses the target model to guide the fine-tuning gradient, whereas PGD directly reduces the prior policy's value at the modified transitions.

For the online state-perturbation comparison, iterative signed-gradient perturbation with $\epsilon = 0.01$ gives $R = 1735 \pm 15$ and $\delta_m = 0.091 \pm 0.006$, while $\epsilon = 0.02$ gives $R = 1540 \pm 24$ and $\delta_m = 0.122 \pm 0.004$. The matched Clean return is 1764 ± 12; SWAAP gives $R = 1409 \pm 275$ and $\delta_m = 0.137 \pm 0.036$. These values are mean $\pm$ sample SD across three Stage 1 runs, averaging the three fine-tuning evaluations within each run. For all methods, δ_m uses the unperturbed current latent state, the executed action, and the observed successor. Online perturbation modifies states at test time, whereas SWAAP modifies fine-tuning data. On this panel, SWAAP produces the largest mean return decrease.

F.9 Trajectory-Consistent Data Poisoning

The Stage 2 formulation in Eq. 5 treats the fine-tuning data as a tupled trajectory data, where each tuple (s_t, a_t, s_{t+1}) can be poisoned by replacing only the next-state target. In sequential trajectory data, however, the same intermediate state appears in two adjacent tuples:

$$(s_t, a_t, s_{t+1}), \qquad (s_{t+1}, a_{t+1}, s_{t+2}).$$

Therefore, replacing s_{t+1} only as the next-state target of the first tuple would make the stored trajectory internally inconsistent. To adapt SWAAP to this setting, we use a trajectory-consistent update: when transition t is selected for poisoning, we replace both occurrences of the intermediate latent state,

$$(s_t, a_t, s_{t+1}) \mapsto (s_t, a_t, \tilde{s}_{t+1}), \qquad (s_{t+1}, a_{t+1}, s_{t+2}) \mapsto (\tilde{s}_{t+1}, a_{t+1}, s_{t+2}).$$

Terminal transitions are excluded so that $t+1$ always belongs to the same trajectory. We use one shared variable for the two occurrences of each selected intermediate state. The matching gradient is computed from the full fine-tuning loss on the sequential poison data:

$$G_{\text{real}} = \nabla_{\psi} \frac{1}{|\tilde{D}|d} \sum_{(\tilde{s},a,\tilde{s}')\in\tilde{D}} \|P_{\psi}(\tilde{s},a) - \tilde{s}'\|_2^2 \Bigg|_{\psi=\psi_0} .$$

The gradient with respect to an editable state includes its effects as both a next-state target and a subsequent current-state input. After each projected update, the same value is written to both occurrences. Here r_p counts selected intermediate states; adjacent selections may affect overlapping tuples.

We evaluate this construction with $\alpha = .9$ and 250 selected intermediate states in a 5,000-transition data ($r_p = .05$), giving 500 changes to stored state fields. We initialize the selected states at their clean values and compute the fine-tuning gradient through both occurrences of each state as above.

Table 9 reports the trajectory-consistent results, with the poisoned states initialized at their clean values. Return reductions range from 10.35% to 95.12% on Humanoid-Walk and from 5.82% to 25.38% on Dog-Run. δ_d ranges from .1032 to .1053 on Humanoid-Walk against clean .0897, and from .0618 to .0626 on Dog-Run against clean .0507. Preserving trajectory continuity therefore retains attack effects with larger prediction residuals than the corresponding clean data.

Table 9: Cumulative test Return for trajectory-consistent SWAAP at $r_p = .05$ and $\alpha = .9$, initialized from clean states. Each row is one Stage 1 run and averages three fine-tuning runs with 100 test episodes each. Humanoid-Walk run 2 and Dog-Run run 1 were used for development; the other runs use the same construction settings.

Task	Stage 1 run	Clean	SWAAP
Humanoid-Walk	1	853.25	764.91
Humanoid-Walk	2	857.02	219.77
Humanoid-Walk	3	860.50	42.01
Dog-Run	1	647.84	483.41
Dog-Run	2	647.73	610.03

G Additional Ablations

G.1 Top-r_p Transition Selection vs. Random Selection

Stage 2 first selects the subset of transitions to poison. Under a fixed poisoning budget r_p, our default heuristic selects the top-r_p transitions with the largest disagreement between pretrained and Stage 1 target predictions. Modifying these transitions aims to provide a stronger fine-tuning signal toward the target than modifying randomly selected transitions.

We test this heuristic on Humanoid-Walk using the same Stage 1 target models and $r_p = .1$, with $\alpha = .9$. Top-r_p selection gives $R = 387 \pm 332$ and $\delta_m = 0.163 \pm 0.052$, while random transition selection gives $R = 853 \pm 8$ and $\delta_m = 0.094 \pm 0.003$. Values are mean $\pm$ sample SD across three Stage 1 runs after averaging three fine-tuning runs per target. The large return gap supports selecting transitions by pretrained–target prediction disagreement.

G.2 MPC Hyperparameter Ablation

Table 10 reports the return of TD-MPC2 under different rollout horizons ($h = 3, 6, 9$) and numbers of samples for both the clean and SWAAP settings. Mean return is lower under SWAAP in all fifteen configurations. MPC performance generally increases with the number of samples and decreases as the rollout horizon grows. The magnitude of the attack effect also depends on the planner: with 512 samples, the Clean-to-SWAAP return changes are $857 \to 387$, $707 \to 137$, and $140 \to 22$ at horizons 3, 6, and 9, respectively. At 64 samples, Clean return is already below 10, leaving little return for the attack to remove.

Table 10: Effect of MPC hyperparameters (`num_samples`, horizon H) on cumulative test Return under Clean and SWAAP in Humanoid-Walk at $r_p = .1$ and $\alpha = .9$. The same fine-tuned models are reevaluated under each planner setting. Values are mean ± sample SD across three independent Stage 1 runs after averaging three final fine-tuning runs per selected target, with 100 test episodes using matched evaluation random seeds per fine-tuned model and planner setting.

Num Samples		Return		
		$h = 3$	$h = 6$	$h = 9$
64	Clean	6 ± 0	8 ± 1	7 ± 1
	SWAAP	5 ± 1	5 ± 2	4 ± 2
128	Clean	187 ± 8	65 ± 6	13 ± 1
	SWAAP	49 ± 41	11 ± 11	5 ± 2
256	Clean	781 ± 4	419 ± 16	51 ± 2
	SWAAP	282 ± 315	42 ± 59	7 ± 6
512	Clean	857 ± 4	707 ± 14	140 ± 10
	SWAAP	387 ± 332	137 ± 199	22 ± 31
1024	Clean	881 ± 3	801 ± 5	285 ± 20
	SWAAP	431 ± 352	179 ± 262	32 ± 48

Table 11: Effect of the Stage 2 poisoning ratio r_p in Humanoid-Walk at $\alpha = .9$. Targets remain those selected at $r_p = .1$; they are not reselected for each ratio. Return is cumulative test return and δ_m is rollout prediction error. Values are mean ± sample SD across three independent Stage 1 runs after averaging three final fine-tuning runs per selected target, with 100 episodes each. The Clean row is the matched clean-fine-tuning control.

r_p	Return	δ_m
clean	857 ± 4	0.09 ± 0.00
0.01	841 ± 30	0.10 ± 0.00
0.05	467 ± 313	0.15 ± 0.04
0.10	387 ± 332	0.16 ± 0.05
0.20	410 ± 308	0.16 ± 0.05
0.30	408 ± 271	0.16 ± 0.04

G.3 Stage 2 Poisoning Ratio r_p

We conduct an ablation study on different r_p ratios in Stage 2, with results in Table 11. Reducing r_p from .1 to .05 raises return from 387 to 467, while $r_p = .01$ gives 841, close to the matched Clean return of 857. Increasing r_p to .2 or .3 gives returns of 410 and 408, respectively. Thus, attack strength depends on the poisoning fraction but does not improve monotonically over this range.

G.4 Larger Fine-Tuning Datasets

We conducted additional experiments on `HumanoidWalk` using fine-tuning dataset sizes ranging from 5,000 to 25,000, with a fixed budget of 500 poisoned transitions. The results are shown in Table 12. Increasing the dataset size reduces the poisoning fraction from 10% to 2%, and the return approaches the matched Clean result at the larger sizes. The mean Clean-minus-SWAAP return gap falls from 469.9 points at 5,000 transitions to 1.9 points at 25,000. Thus, the effect weakens substantially when the number of poisoned transitions is kept fixed.

We also keep the poisoning fraction fixed at $r_p = .1$ when increasing the fine-tuning dataset to 25,000 transitions, allowing 2,500 poisoned transitions. The mean return reduction is $49.01 \pm 38.95\%$ on Humanoid-Walk and $38.96 \pm 15.81\%$ on Dog-Run across three Stage 1 runs, each averaging three fine-tuning runs with 100 test episodes per model. Both use $\alpha = .9$ and ten epochs. The larger dataset requires 980 updates rather than 200; the fixed-fraction result therefore uses more poisoned transitions and more victim updates than the 5,000-transition experiment.

Table 12: Effect of fine-tuning dataset size in Humanoid-Walk at $\alpha = .9$, with exactly 500 poisoned rows at every size. Return is SWAAP cumulative test return and δ_m is rollout prediction error. Values are mean $\pm$ sample SD across three independent Stage 1 runs after averaging three fine-tuning runs per selected target; each fine-tuned model uses 100 episodes with paired evaluation random seeds. The number of epochs remains ten, so optimizer updates increase with fine-tuning dataset size. The 5,000-row condition reuses the main experiment.

Fine-tuning Dataset Size	**Return**	δ_m
5,000	387 ± 332	0.163 ± 0.052
10,000	799 ± 35	0.099 ± 0.004
15,000	837 ± 11	0.095 ± 0.002
20,000	850 ± 8	0.093 ± 0.002
25,000	858 ± 3	0.092 ± 0.001

H Extensions

H.1 DINO-WM Push-T Results

We adapted our method to DINO-WM for goal-conditioned tasks [Zhou et al., 2025]. Stage 1 uses a hierarchical goal-conditioned RL policy with a subgoal module as its surrogate, while the deployed agent uses MPC with the cross-entropy method (CEM). Table 14 gives the model and planning settings.

The fine-tuning dataset contains 320 history-dependent transitions from 40 Original-policy trajectories, with a separate 80-transition reference dataset for gradient matching. The attacker selects 32 successors from 80 transitions it can modify, giving $r_p = .1$ over the full fine-tuning dataset. The remaining 240 transitions stay clean. The matched fine-tuning gradient includes all 320 transitions. Each successor has 196 tokens with 394 state features. Action features and input histories remain fixed. We use 5,000 gradient steps of size 10 and a clean-centered coordinatewise bound $\epsilon = 1$. For these high-dimensional features, each residual norm in the prediction-error regularizer is divided by $\sqrt{196 \times 394}$, so the squared regularizer is scaled by $1/(196 \times 394)$ relative to Eq. 5. Fine-tuning updates only the predictor using AdamW, learning rate 2×10^{-5}, batch size 16 and ten epochs (200 updates). Fitting uses bfloat16, and planning uses float32.

Table 13 reports the final evaluation on the same 20 goals. Direct Model Poisoning replaces only the predictor and reduces success from $16/20$ to $1/20$. SWAAP uses one fixed target and poisoned dataset for each α, with three fine-tuning runs paired with Clean. It reduces success by 8.33 ± 2.89 and 6.67 ± 2.89 percentage points at $\alpha = 0$ and .9, respectively. The average seven-dimensional state distance improves slightly. This secondary measure includes velocity, whereas task success depends on position and angle. The success losses are modest and occur on three distinct goals across the fitting runs. On the development and validation panels, mean success losses are -3.33 and 3.33 percentage points for $\alpha = 0$, and 3.33 and 0 for $\alpha = .9$. The final panel was fixed before construction. The measured effect therefore depends on the goal panel as well as the fine-tuning run.

Table 13: DINO-WM Push-T results at $r_p = .1$ on the same 20 test goals. Success rates are mean $\pm$ sample SD across three fine-tuning runs of one fixed target and dataset, paired with Clean; Original and Direct Model Poisoning each use one model. Direct replaces only the predictor. δ_d is measured once per dataset using float32 pretrained predictions over all 320 history-dependent transitions and the 196×394 successor-state features. The Clean value, $\delta_{d,\mathrm{ref}} = .089085$, is the prediction error on unmodified data before fine-tuning.

Method	Success rate (%)	δ_d
Original	80.00	
Direct Model Poisoning	5.00	
Clean fine-tuning	83.33 ± 2.89	.089085
SWAAP, $\alpha = 0$	75.00 ± 0.00	.089170
SWAAP, $\alpha = .9$	76.67 ± 2.89	.089083

Table 14: Key hyperparameters used in the DINO-WM Push-T experiment.

Hyperparameter	**Value**	**Description**
obs_shape	(3, 224, 224)	Shape of pixel observation s.
latent_shape	(196, 394)	Shape of latent state $z = \mathrm{enc}(s)$.
action_dim	10	Dimension of a group of five two-dimensional actions.
num_hist	3	Historical latent states used to predict the next state.
H	5	Planning horizon.
num_samples	100	Candidate trajectories per CEM iteration.
opt_steps	10	CEM optimization iterations per planning step.

Table 15: Gray-box transfer from the MW-Soccer surrogate to the multi-task victim with $r_p = .1$, $\alpha = 0$, and a $.0045$ coordinate-relative raw-observation bound. Cumulative Return and rollout prediction error δ_m are mean $\pm$ sample SD over 100 paired test episodes. Each learning rate uses one Clean and one SWAAP fine-tuning run on Soccer data; the Task column specifies downstream evaluation. Original is the pretrained victim.

Task	Learning rate	Setting	Return	δ_m
MW-Soccer		Original	1407.8 ± 355.9	0.0999 ± 0.0314
MW-Soccer	2.5×10^{-5}	Clean	1371.5 ± 341.8	0.0997 ± 0.0365
MW-Soccer	2.5×10^{-5}	SWAAP	1247.0 ± 427.4	0.1026 ± 0.0396
MW-Soccer	5×10^{-5}	Clean	1334.2 ± 362.7	0.1000 ± 0.0367
MW-Soccer	5×10^{-5}	SWAAP	1264.9 ± 390.6	0.1001 ± 0.0349
MW-Push		Original	1233.8 ± 618.7	0.1385 ± 0.0890
MW-Push	2.5×10^{-5}	Clean	1327.4 ± 558.9	0.1220 ± 0.0784
MW-Push	2.5×10^{-5}	SWAAP	1184.3 ± 507.8	0.1035 ± 0.0669
MW-Push	5×10^{-5}	Clean	1406.6 ± 546.0	0.1133 ± 0.0799
MW-Push	5×10^{-5}	SWAAP	1155.6 ± 460.3	0.0937 ± 0.0487

For pre-training detection, we use the pretrained model in float32 and the residual-CUSUM procedure from Appendix F.5.1. Each trajectory contributes eight ordered prediction residuals from histories of length three. Separate sets of 200, 200 and 400 clean trajectories provide residual statistics, thresholds and independent testing. Both poisons have AUC $.4945$ and five alarms among all 40 trajectories at each nominal $5\%/10\%$ threshold, exactly matching their unmodified versions. None of the seven trajectories containing edited successors acquires an alarm. The independent Clean false-positive rates are 8.5% and 12.75%.

H.2 Gray-Box Attack on TD-MPC2 Multi-Task World Model

We conducted a gray-box experiment in which the attacker has access to a surrogate world model trained on MW-Soccer, while the victim is a TD-MPC2 model trained across 80 tasks. The attacker performs target selection and poison construction using the single-task surrogate, then supplies the poisoned dataset to the victim for fine-tuning.

The attack modifies raw successor observations and differentiates through the frozen surrogate encoder. The victim independently encodes the raw data with its own task-dependent encoder. We use $r_p = .1$ and $\alpha = 0$, changing 500 of 5,000 Soccer transitions. The coordinatewise constraint is $|\tilde{s}'_{i,j} - s'_{i,j}| \leq .0045|s'_{i,j}|$, with zero-valued coordinates fixed. Earlier multi-task evaluations informed development of the attack. The final target and perturbation magnitude were selected using surrogate measurements and fixed before this candidate was evaluated on the victim. The victim updates only its dynamics on the Soccer data, using Soccer task conditioning, ten epochs, batch size 256 and learning rates 2.5×10^{-5} and 5×10^{-5}. Each downstream task is evaluated using its own task conditioning.

Table 15 reports matched comparisons on Soccer and Push. Soccer return decreases by 9.08% and 5.20% relative to Clean fine-tuning at the two learning rates; Push decreases by 10.78% and 17.84%. The table also gives pretrained Original performance, separating changes from clean adaptation and changes from poisoning.

We also evaluate the same fitted models on all 80 tasks using 50 paired episodes per task. At the lower and higher learning rates, reductions reach at least 5% on 16 and 12 tasks, respectively, and at least 10% on eight tasks at each rate. Returns increase by at least 5% on 19 and 25 tasks, respectively. Affected tasks include locomotion, pushing, pulling, hammering and pick-and-place. Effects in these fixed adapted models are selective, with median reductions of -0.32% and -0.41%. The full signed task results are retained in the accompanying results data.

Data deviation is computed separately with each encoder. The surrogate gives $\delta_d = .05149$ against a clean reference of $.04714$; the victim under Soccer conditioning gives $.04734$ against $.04563$. A victim-reference residual-CUSUM detector uses 300 clean trajectories for statistics, 500 for thresholds and 1,000 for independent testing. It alarms on $1/50$ and $6/50$ poisoned trajectories at nominal $5\%/10\%$ thresholds, compared with independent Clean false-positive rates of $5.5\%/10.4\%$. Its AUC is $.639$, with standardized pAUC_5 and pAUC_{10} of $.487$ and $.484$. These detector measurements screen the single Soccer dataset used for both fine-tuning schedules.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's main contribution as a two-stage data-poisoning framework for learned world models, with Stage 1 target-model identification and Stage 2 stealth-constrained data realization. They also describe the evaluated scope, including continuous-control benchmarks and stealth evaluations across pre-training detection, robust training, and model-level deviation.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses several limitations and scope restrictions, including the main white-box threat assumption, the preliminary nature of the gray-box transfer result, the Stage 1–Stage 2 target-realization bottleneck, and the need to evaluate broader world-model backbones and larger-scale foundation models. These limitations are discussed in the threat model, experimental ablations/generalization section, and conclusion/future-work discussion.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The paper states the transition-gradient theorem in the main text with its differentiability and finite-horizon stochastic-transition assumptions, and provides the full derivation in Appendix D.1. The appendix also includes the stochastic, finite-horizon, and deterministic specializations used by the method.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper describes the victim agent, fine-tuning setup, poisoning budget, evaluated benchmarks, baselines, deviation metrics, number of runs, and defense settings used for the main results. Additional implementation details, hyperparameters, compute costs, and extended ablations are provided in the appendix, and we will release repository-style code and scripts to reproduce the main experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.

(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We will provide anonymized repository-style code in the supplemental material, including environment setup, data-generation/preprocessing instructions, and scripts for reproducing the main SWAAP experiments and baselines. The experiments use standard simulated benchmarks and generated fine-tuning data; instructions will specify how to regenerate the required clean, poisoned, and evaluation trajectories.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The main text specifies the victim agent, benchmarks, pretraining and fine-tuning sizes, poisoning budget, planning horizon, candidate rollouts, metrics, baselines, and number of runs. Additional hyperparameters, optimizer choices, defense settings, compute details, and extended experimental procedures are provided in the appendix and will be included in the reproducibility code.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.

- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Table 1 reports mean $\pm$ standard deviation for the main experimental results. The experimental setup states that each table entry aggregates 10 independent runs, where each run is itself averaged over approximately 5,000 test transitions using 10 episodes for 500-step DMControl tasks and 50 episodes for 100-step tasks.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The appendix reports the hardware environment and approximate runtime for the main experimental pipeline, including compute for Stage 1 target-model optimization, Stage 2 data poisoning, fine-tuning, and evaluation. We also disclose the estimated total compute needed to reproduce the reported experiments.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: The research is conducted in a simulated world-model/continuous-control setting and is presented as an adversarial robustness evaluation intended to identify and mitigate vulnerabilities in world-model adaptation pipelines. The paper preserves anonymity and does not use human-subject data, private data, or sensitive personal information.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The paper discusses positive impacts in terms of identifying vulnerabilities and motivating robustness methods for learned world-model adaptation pipelines. It also discusses negative impacts and misuse risk because the method is an attack; the work is evaluated in simulated environments and framed as a benchmark for improving defenses, with detection and robust-training evaluations included as mitigation-oriented analysis.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk pretrained generative models, scraped datasets, or real-world sensitive data. The planned release consists of research code and scripts for simulated continuous-control benchmarks, so the specific safeguard question for high-risk model or dataset release is not applicable.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper cites the original works for the existing benchmarks, environments, and world-model agents used in the experiments, including TD-MPC2, DMControl, MyoSuite, MetaWorld, ManiSkill2, and DINO-WM. We will include the relevant asset versions, URLs, and license information in the released repository and ensure that any redistributed code or generated data respects the original terms of use.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper introduces new research code, poisoning scripts, generated fine-tuning data, and experimental artifacts for SWAAP. These assets will be released with documentation describing setup, usage, reproduction commands, generated-data format, limitations, and licensing, with anonymization preserved at submission time.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.

- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, human-subject experiments, participant instructions, or compensation. All experiments are conducted in simulated continuous-control environments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, human-subject studies, or data collected from human participants. All experiments are conducted in simulated continuous-control environments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were not an important, original, or non-standard component of the core research methodology. Their use for manuscript preparation, formatting, and software-release support is described in Appendix A. The authors checked and approved the technical claims and released artifacts.

    Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.