stable-worldmodel: A Platform for Reproducible World Modeling Research and Evaluation

Lucas Maes[*1] **Quentin Le Lidec**[*2] **Luiz Facury**[3] **Nassim Massaudi**[4] **Ayush Chaurasia**[5]
Francesco Capuano[6] **Richard Gao**[7] **Taj Gillin**[7] **Dan Haramati**[7] **Damien Scieur**[1]
Yann LeCun[2] **Randall Balestriero**[7]

[1]Mila & Université de Montréal [2]New York University [3]Universidade Federal de Minas Gerais
[4]Independent Researcher [5]LanceDB [6]University of Oxford [7]Brown University

Abstract

World models are central to building agents that can reason, plan, and generalize beyond their training data. However, research on world models is currently fragmented, with disparate codebases, data pipelines, and evaluation protocols hindering reproducibility and fair comparison. Current practice is further limited by three key bottlenecks: fragile one-off codebases, slow video data loading, and the lack of standardized generalization benchmarks. We present `stable-worldmodel` (`swm`), an open-source platform for standardized and reproducible world modeling research and evaluation. It delivers (1) a high-performance Lance-based data layer with native support and conversion tools for MP4, HDF5, and LeRobot datasets, (2) clean, well-tested implementations of modern world model baselines and planning solvers, and (3) a broad suite of environments and tasks extended with controllable visual, geometric, and physical factors of variation for systematic in-silico evaluation of dynamics understanding, control performance, representation quality, and out-of-distribution generalization. By unifying the full pipeline under a single, scalable framework, `swm` dramatically reduces research overhead and accelerates trustworthy progress toward reliable world models. Code available here.

1 Introduction

The idea of using predictive models to guide decision-making dates back to the 1960s-1970s from the control theory community [1, 2]. Most of these approaches relied on analytical, closed-form models or hand-crafted simulators to predict the outcome of actions (control variables) [3, 4]. Yet, recent advances in deep learning have instead learned to build these predictive models directly from raw data, leveraging neural networks [5–9], an approach more commonly referred to as *world models*. These world models enable the capture of far more complex real-world phenomena [10, 11] without deriving dynamics equations explicitly. However, as for any learned system, strengths and weaknesses audits before real-world deployment are paramount. Yet, despite exciting progress, the world modeling community still lacks reliable baselines,

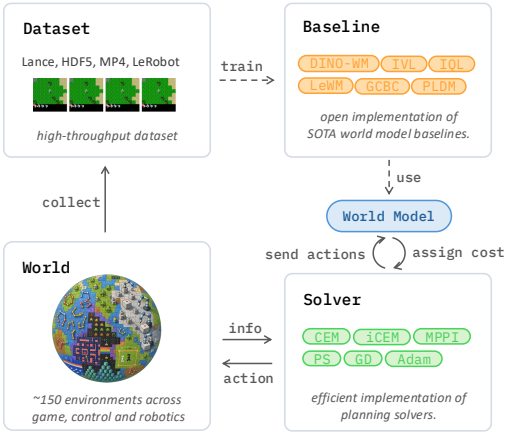
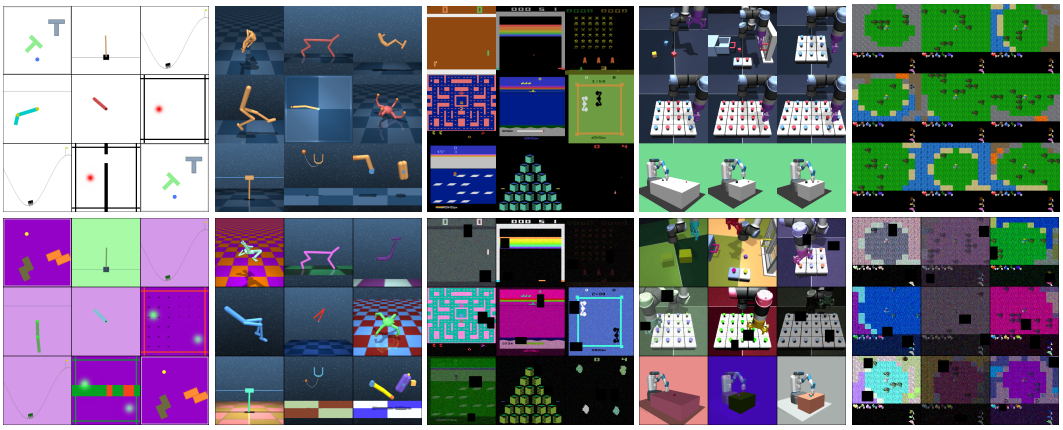


Figure 1: Overview of `stable-worldmodel`: data is efficiently collected from a world and used to train world models via provided baselines, then leveraged by solvers for control.

* Equal contribution. Correspondence to `lucas.maes@mila.quebec`



(a) Classic Control (b) MuJoCo (c) Arcade Game (d) Robotics (e) Open-World Game

Figure 2: Environment families supported by `swm`. **Top row:** default (unperturbed) renderings of each environment. **Bottom row:** all visual factors of variation (e.g., agent, object, scene, geometry, lighting) jointly perturbed. Dynamic physical parameters (e.g., mass, density, gravity, or friction) can also be modified, but are omitted here as they are not visible in a single frame.

standardized benchmarks, and reproducible evaluation protocols. Most research relies on custom, often fragile codebases, forcing researchers to repeatedly re-implement the same algorithms. As the history of computer science has repeatedly shown, this approach is a recipe for hidden bugs, inconsistent results, leading to reduced credibility and trustworthiness in the results reported in publications [12–14].

To address these challenges, we present `stable-worldmodel` (`swm`), an open-source platform for reproducible world modeling research and evaluation. Built on PyTorch [15] and Gymnasium [16], `swm` provides a complete, modular test-bed that supports researchers across the entire world model pipeline: from dataset collection and training utilities to comprehensive evaluation (including control, representation probing, out-of-distribution, and zero-shot generalization). It enables systematic assessment of new algorithms, clear identification of *in-silico* model limitations, and fair comparisons against strong baselines.

Our contributions are the following:

- A high-performance Lance-based data layer with native support and conversion tools for MP4, HDF5, and LeRobot datasets, eliminating I/O bottlenecks in multimodal world model training.
- Clean, well-tested, and reproducible implementations of modern world model baselines (DINO-WM, LeWorldModel, PLDM, TD-MPC2, etc.) together with efficient and reliable implementations of planning solvers (CEM, MPPI, GD, etc.).
- A diverse benchmarking suite spanning classic control, MuJoCo, Atari, robotics, and open-world environments, augmented with systematic controllable factors of variation (visual, geometric, and physical) enabling standardized zero-shot generalization and robustness evaluation.

2 Background: the World Model Pipeline

In this section, we provide the necessary background on world models and control, and highlight key challenges that arise in their design, training, and evaluation.

Training World Models. A world model is mostly the association of two core components, an encoder and a predictor. The encoder $\mathcal{E} : \mathbb{R}^n \to \mathbb{R}^D$ maps a (possibly partial) observation $\mathbf{o}_t \in \mathbb{R}^n$ to a latent state $\mathbf{s}_t \in \mathbb{R}^D$. The predictor $\mathcal{P} : \mathbb{R}^D \times \mathbb{R}^A \to \mathbb{R}^D$ learns the dynamics by predicting the next state $\mathbf{s}_{t+1}$ given the current state $\mathbf{s}_t$ and an action $\mathbf{a}_t$. In practice, learning such representations from high-dimensional sensory inputs, such as videos, has become ubiquitous, yet it requires compressing

observations into compact latent states that retain only task-relevant features. This has led to a wide range of representation learning strategies, including reconstruction-based objectives [17, 8, 11] or JEPAs [18–21].

World modeling requires latent representations that faithfully capture a system's underlying dynamics, since accurate long-horizon prediction and planning depend on understanding how states evolve over time. This places strong demands on the representations: they must support rich temporal reasoning and generalize robustly to out-of-distribution scenarios.

Planning with World Models. Once learned, a world model can be used to determine a sequence of actions that drives the system toward a desired goal state $\boldsymbol{s}_g \in \mathbb{R}^D$. This problem can be framed as optimal control, where the objective is to find a policy π which minimizes the expected cumulative cost induced by the learned dynamics. Formally, given an initial state $\boldsymbol{s}_0$, we seek a mapping $\pi : \mathbb{R}^D \to \mathbb{R}^A$ such that

$$\pi^* = \arg\min_{\pi} \sum_t c(\boldsymbol{s}_t, \boldsymbol{a}_t) \quad \text{s.t.} \quad \boldsymbol{s}_{t+1} = \mathcal{P}(\boldsymbol{s}_t, \boldsymbol{a}_t), \quad \boldsymbol{a}_t = \pi(\boldsymbol{s}_t), \tag{1}$$

where $\mathcal{P}$ denotes the world model predictor and c is a task-dependent stage cost measuring progress toward the goal $\boldsymbol{s}_g$.

Approaches to solving equation 1 typically fall into two categories. Reinforcement Learning (RL) learns the policy π offline (or online) by optimizing it over many trajectories [7, 17], while Model Predictive Control (MPC) solves a finite-horizon (T steps) version of this problem at test time by directly optimizing predicted trajectories over action sequences [22, 21]. Unlike RL, MPC leverages the world model directly as part of the agent. For the rest of this work, we will rely on MPC to solve control, yet our platform is also compatible with RL as both approaches ultimately aim to solve the same underlying control problem.

3 `stable-worldmodel`: A Unified Framework for World Model Research

We introduce `stable-worldmodel` (Fig. 1), an open-source platform for reproducible world model research and evaluation that supports researchers across the entire pipeline. As detailed in the following subsections, `swm` is motivated by key challenges in current practice, including fragmented implementations, data handling bottlenecks, and limitations in evaluation protocols. `swm` is entirely open-source and designed to integrate smoothly with existing code bases.

3.1 Challenges in World Modeling Research

We highlight key challenges across the world-modeling pipeline that motivate the need for a shared, standardized platform enabling efficient, reproducible, and rigorous evaluation and iteration.

Reproducibility and Implementation Fragmentation. As depicted before, the plurality of approaches and motivations to tackle the world modeling task has led most recent works to operate in silos: each lab designs its own end-to-end pipeline for data collection, model training, and evaluation. In addition, these works routinely re-implement the same baselines, environments, and core algorithms from scratch. For example, the Cross-Entropy Method (CEM) [23] planner has been independently re-implemented (with varying degrees of fidelity) in at least five recent papers, including TDMPC [8], PLDM [19], DINO-WM [22], LeWM [21], and V-JEPA2 [10]. Repeatedly re-implementing the same components across different codebases often introduces subtle inconsistencies that undermine fair comparison and reproducibility [12, 14]. The resulting fragmentation reduces the trustworthiness of reported results and makes it difficult to isolate whether performance gains stem from genuine methodological advances or from implementation differences.

Data Bottlenecks in WMs learning. World models rely on an encoder $\mathcal{E}$ to infer latent states $\boldsymbol{s}_t$ from raw observations $\boldsymbol{o}_t$. Unlike traditional pipelines, training such models requires loading multimodal data as contiguous temporal blocks, including video frames, actions, proprioception, and other sensor streams. This requirement creates a fundamental trade-off between random access efficiency and I/O throughput. Storing trajectories as individual frames (standard in computer vision)

enables fast random sampling but incurs prohibitive I/O overhead and storage costs due to the high redundancy of file header decoding. Conversely, encoding trajectories as compressed video clips (e.g., MP4) drastically reduces disk footprint, yet severely degrades random access: retrieving a later frame often requires decoding all preceding frames in the clip. Consequently, neither approach scales effectively. Data loading is a real bottleneck, leading to GPU starvation, as loading fails to keep the accelerator utilized.

Evaluating Robust World Models. Current world model research mostly depends on simulator evaluation because transferring models and controllers to real environments is expensive, time-consuming, and technically demanding. As a result, Gym-style simulation benchmarks have become the dominant proxy for measuring generalization and control performance [16]. However, standard benchmarks often evaluate models under conditions close to the training distribution, making it difficult to determine whether a world model has learned reusable environment dynamics or merely exploitable correlations in the data. This distinction is central to world modeling: a world model is not itself a policy, but a learned representation of how observations, states, and actions evolve over time, which can then be used by MPC, model-based RL, or other downstream planners. Recent methods such as DINO-WM [22], PLDM [19], LeWM [21], TD-MPC2 [8], and Genie [24] illustrate the diversity of architectures and objectives used to learn such predictive structure. Yet predictive accuracy or task success in-distribution does not necessarily imply that the learned latent dynamics are temporally stable, intervention-robust, or useful for counterfactual reasoning and long-horizon planning. Recent analyses of learned physical systems further show that high-capacity sequence models can fit trajectories accurately while failing to recover the local dynamical laws that generated them [25]. This limitation is especially problematic because many important dynamical modes, such as contact interactions, occlusions, or changes in physical parameters, only appear under specific behaviors or environment configurations that may be absent from the training data. Consequently, a model may appear successful on standard benchmarks while still harboring fundamental misunderstandings of the underlying dynamics. Zero-shot and out-of-distribution evaluation help expose such failures by testing whether learned dynamics remain useful under controlled changes in appearance, geometry, object properties, memory requirements, or physical parameters.

3.2 Design Principles and Interface

The core philosophy behind `swm` is to impose as few restrictions as possible on the user's model architecture and training code, while providing strong standardization for data collection, evaluation, and control. Researchers typically have their own preferred training frameworks and model designs; forcing a particular training loop would limit adoption. In contrast, data collection, environment interaction, and evaluation protocols benefit greatly from standardization, enabling fair and reproducible comparisons.

To achieve this balance, `swm` is built around three minimal yet powerful abstractions (Fig. 1):

- `World`: A unified environment wrapper that supports data collection, policy execution, and evaluation across diverse simulators (Gymnasium-compatible). It handles vectorized execution, rendering, and controllable intervention on the environment's visual, geometric, and physical properties, which we call *factors of variation* (FoV).
- `Policy`: A simple interface that maps observations (or latent states) to actions. This includes random policies, expert policies (e.g., SAC), and learned policies from RL or MPC. In particular, `MPCPolicy` wraps any world model and planner solver: at each timestep it encodes the current observation into a latent state and delegates action selection to the underlying solver by solving the finite-horizon optimal control problem (Equation 1).
- `Solver`: A self-contained and robust implementation of various single-shooting planning algorithms for MPC. Supported solvers include the Cross-Entropy Method (CEM), Model Predictive Path Integral (MPPI), Gradient Descent, Projected Gradient Descent, and many more. Each solver simply expects their associated world model to implement a `get_cost` method and use that signal to return an optimized finite-horizon action sequence. All solvers are extensively tested and benchmarked for numerical stability and performance.

Overall, our non-invasive design enables a clean, reproducible research loop with minimal boilerplate, as illustrated in Algorithm 1. Furthermore, by separating model training from the evaluation infras-

Format	Throughput (sample/sec)	
	w/o caching	w/ caching
HDF5 (local)	1,416	1,474
HDF5 (S3)	9	757
Lance (local)	4,815	4,431
Lance (S3)	3,184	3,253
Video (local)	1,331	1,348

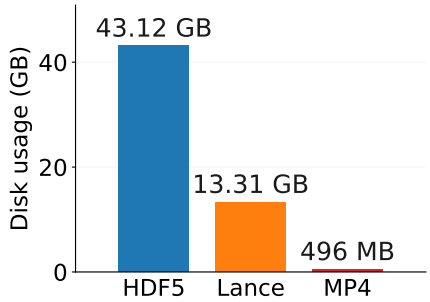


Figure 3: Performance comparison of different data formats for a dataset from the Push-T environment. **(Left)** Data loading throughput (samples/sec) with and without caching, for both local storage and remote (S3) streaming. **(Right)** Disk storage usage. Results demonstrate that Lance is the most efficient format in terms of throughput while remaining a good trade-off in compression for storage.

tructure, `swm` lets researchers focus on algorithmic innovation while benefiting from battle-tested data pipelines, planning solvers, and standardized benchmarks. New environments can be added with minimal effort, as long as they follow the Gymnasium API.

```python
import stable_worldmodel as swm
# 1] Create the world
world = swm.World(
    env_name="swm/PushT-v1", num_envs=8, max_episode_steps=1000)
# 2] Collect data
world.set_policy(swm.policy.RandomPolicy(seed=42))
world.collect(dataset_path="pusht.lance", episodes=5000, seed=42)
# 3] Train/Load your world model (user-defined)
model = MyWorldModel(...) # Any PyTorch model
# 4] Evaluate with planning
planner = swm.solver.CEMSolver(model, ...)
wm_policy = swm.policy.MPCPolicy(planner)
world.set_policy(wm_policy)
metrics = world.evaluate(
    episodes=100, seed=0, video="videos/",
    options={'variation': ['agent.size', 'background.color']})
print(metrics["success_rate"])
```

Algorithm 1: Typical `stable-worldmodel` loop illustrating the full pipeline: **(1)** instantiating a world composed of 8 Push-T environments **(2)** collecting a dataset using a given policy (here random, but expert policies are also supported), **(3)** training or loading a user-defined world model, **(4)** creating a planner solver and wrapping it into an MPC policy, which and run evaluation on a modified environment with a random agent size and background color. The same interface works with any supported solver and environment.

3.3 System components

Here, we describe the concrete choices made in `swm` to directly address the challenges outlined in Sec. 3.1. We begin with the data layer, detailing the decisions that enable efficient collection and loading of multimodal trajectories. We then present the baseline world models and planning solvers currently implemented in the platform. Finally, we introduce `swm`'s core contribution: its evaluation testbed. This includes a broad suite of supported environments together with a rich set of customizable factors of variation (visual, geometric, and physical) that can be applied on-the-fly to any environment.

Data Layer. `swm` provides a flexible data collection pipeline supporting both offline and online regimes. Users can generate trajectories with random policies for broad exploration, pre-trained expert policies, policies loaded from model checkpoints, or any mixture thereof. This design makes it

straightforward to study the influence of data composition on world model performance. To perform online world model training as in TD-MPC2 [8], the platform natively supports iterative alternation between data collection and model training or evaluation by simply repeating the steps of Alg. 1. All collections are performed through the unified `World` interface and can be combined on-the-fly with controllable factors of variation (FoV) for visual, geometric, or physical perturbations (Fig. 2).

As discussed in Sec. 3.1, efficient loading of multimodal, temporally contiguous trajectories is a major bottleneck in world model training. To address this, `swm` adopts Lance [26] as its primary storage format. Lance is a modern, columnar, ML-optimized format that offers fast random access, high compression ratios, zero-copy operations, native versioning, and seamless streaming from cloud object stores. However, for maximum compatibility, `swm` also supports common alternatives (MP4, HDF5, and LeRobot [27]) and provides one-click conversion tools to Lance. This allows researchers to immediately use existing datasets while benefiting from standardized, high-performance storage. In particular, LeRobot datasets include a large and growing collection of real-robot trajectories, which are seamlessly integrated via an adapter and can be automatically converted to Lance for optimal throughput. Benchmarks on Push-T (Fig. 3) and Two-Room (Fig. 8) validate that Lance achieves the highest loading throughput, outperforming locally stored HDF5, MP4, and LeRobot, and even surpassing remote HDF5 when streaming online from S3.

Control Layer. As noted in Sec. 3.1, commonly used solvers have been independently reimplemented across many recent world model papers. `swm` addresses this by providing a collection of planning solvers, easily pluggable with any trained world model through an `MPCPolicy` interface. At every environment step, the active solver receives the current latent state and uses the world model to evaluate candidate action sequences by rolling out their trajectories over a planning horizon. It then returns an optimized action sequence, of which the K first actions are executed before replanning at the next step. The implemented solvers cover two broad families of approaches. Sampling-based and Gradient-based methods. Sampling methods include Predictive Sampling [28], Cross-Entropy Method (CEM) [23], improved CEM (iCEM) [29], and Model Predictive Path Integral (MPPI) [30]. They iteratively sample, evaluate, and refine populations of candidate action sequences without requiring any information on the model (e.g., gradient). Gradient-based methods, including Gradient Descent, Projected Gradient Descent [31], and GRASP [32], instead exploit differentiability of the predictor to optimize actions through back-propagation. All solvers are tested and validated end-to-end: when paired with their original world models, our implementations reproduce the planning success rates reported in DINO-WM [22] and PLDM [19] (see Sec. 4.1), confirming that the unified interface introduces no performance regression. Pseudocode for all solvers is provided in App. F.

Model Layer. `swm` implements multiple baselines that mainly span two paradigms: goal-conditioned reinforcement learning (GCRL) and latent world models with test-time planning. Both instantiate control through a subclass of the common `Policy` interface. Concretely, GCRL methods directly parameterize a `FeedForwardPolicy` that maps observations (and goals) to actions, whereas world model approaches wrap a learned dynamics model inside an `MPCPolicy`, which selects actions by solving the finite-horizon planning problem described in Eq. 1 at each timestep. As described in Sec. 3.2, all methods share a common data management and evaluation loop, removing any ambiguity from comparisons. On the GCRL side, goal-conditioned behavioral cloning (GCBC [33]) is a supervised learning approach that trains policies on expert trajectories. Goal-conditioned implicit Q-learning methods (e.g., GCIVL and GCIQL [34]) instead learn Q functions via expectile regression and extract policies through advantage-weighted updates. On the planning side, world models follow the structure described in Sec. 2, consisting primarily of an encoder $\mathcal{E}$ and a predictor $\mathcal{P}$. At test time, these models are used within an `MPCPolicy`, which encodes the current observation into a latent state and optimizes actions with respect to a goal observation by solving the planning objective. The different world model baselines share this abstraction and differ mainly in their training objectives and architectural choices. DINO-WM [22] freezes a pretrained DINOv2 encoder [35] and learns a ViT [36] predictor over spatial patch features, serving as a baseline for latent world models from foundation models. Planning with a Latent Dynamics Model (PLDM) [19] learns latent representations and dynamics jointly from offline trajectories using a JEPA-style objective, with additional regularization to stabilize training. LeWorldModel (LeWM) [21] is also a JEPA-based world model that simplifies the training objective while maintaining stable learning, and is used here as the fastest latent world model baseline. Finally, we include TD-MPC2 [8], a decoder-free world model that performs local trajectory optimization guided by discrete reward and value predictions.

3.4 Benchmarking World Models

`swm` supports online evaluation with boundary conditions sampled either randomly from the state space or from selected trajectories, following [22, 19, 21]; see App. E for details. We report success rate as the primary metric for control and planning performance.

This protocol applies across environments spanning multiple regimes (Fig. 2): low-dimensional and 2D control (CartPole [37], PushT [38]), 3D visual environments (OGBench [39]), discrete domains (Atari [40]), continuous control (MuJoCo [41]), and partial observability (Craftax [42]). The suite spans reactive dynamics (Atari, MuJoCo), long-horizon planning (OGBench manipulation, Craftax), and varied action and state spaces.

To probe robustness and zero-shot generalization, we introduce controlled factors of variation (FoV) (Fig. 2) targeting visual factors (colors, lighting, textures, occlusions) and physical properties (masses, friction, dynamics). When simulator internals are inaccessible (e.g., Atari ROMs), lightweight visual wrappers can be applied at the observation level (Fig. 6), enabling systematic measurement of OOD transfer, continual learning, or generalization. More details can be found in App. C

4 Case Study: Planning Performance and Zero-Shot Evaluation of WMs

We showcase the practical utility of `swm` on the Push-T manipulation benchmark, a standard testbed for evaluating world models. Using the platform's unified evaluation pipeline, we demonstrate how `swm` enables seamless assessment of planning performance and zero-shot generalization across diverse settings with minimal additional code. These experiments highlight common failure modes in existing models while illustrating how `swm` facilitates systematic diagnosis and comparison. Detailed experimental setup and additional results are provided in App. I and App. J.

4.1 In-distribution Planning Performance

We first report the in-distribution planning performance of representative state-of-the-art world models on Push-T: TD-MPC2 [8], GCBC [33], LeWorldModel [21], PLDM [19], and DINO-WM [22]. All models were trained on the same expert dataset used in DINO-WM and evaluated under identical planning configurations. Tab. 1 summarizes their success rates. We recover performance values consistent with those originally reported for baselines evaluated on Push-T. However, we observed that TD-MPC2 performs poorly in an offline setting; we conjecture that this stems from generating OOD actions, fooling the predictor (see Fig. 13 in App. J for a direct visualization of this drift). We verify our TD-MPC2 implementation by benchmarking it online on the DeepmindControl tasks used in the original paper and compare against a Soft-Actor-Critic (SAC) baseline from Stable-Baselines3 (Tab. 5).

Table 1: Baseline comparison.

Method	Push-T	OGB-Cube
TD-MPC2	12	4
GCBC	75	84
LeWM	94	72
PLDM	78	62
DINO-WM	92	86

4.2 World Models Are Still Brittle Under Distribution Shift

We assess the robustness of world models under distribution shift through progressive shifts and controlled visual perturbations.

We first consider progressive distribution shift by performing planning on expert training data, held-out expert validation data, random-policy trajectories, and finally random-policy trajectories with all `swm` factors of variation (color, shape, size, background, etc.). For each of the four settings, we run 256 trajectories and plot the distribution of trajectory-level prediction MSE for successful and failed planning (Fig. 4 and 10). Prediction error correlates poorly with planning success for both models. While more OOD settings produce higher prediction error, the success and failure distributions largely overlap even under strong distribution shifts. This suggests that out-of-distribution inputs, rather than raw prediction error magnitude, are the primary driver of planning failures.

To more directly isolate the effect of distribution shift, we next introduce targeted visual perturbations using `swm`'s factors-of-variation interface. As shown in Tab. 5b, planning success rates drop sharply for most models even under mild perturbations, confirming that current world models remain brittle

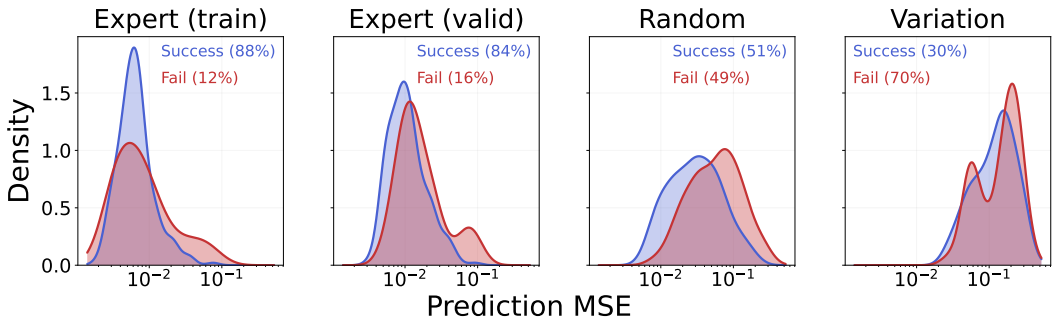
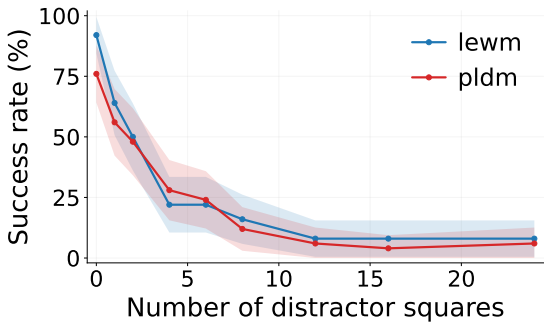


Figure 4: Distribution of trajectory-level prediction MSE for successful (blue) and failed (red) plans on Push-T using LeWM, across four levels of increasing distribution shift. From left to right: expert training trajectories, held-out expert validation trajectories, random-policy trajectories, and random-policy trajectories with full variations (color, shape, size, background, etc.). Prediction error does not seem to correlate with SR. The same qualitative result holds for PLDM (see Fig. 10).



(a) SR evolution wrt visual distractors on Push-T.

FoV	Entity	SR % (↑)		
		LeWM	PLDM	DINO-WM
None		**50.8**	**50.8**	**20.0**
Color	Agent	12.0	8.0	18.0
	Block	22.0	18.0	18.0
	Canvas	6.0	6.0	10.0
Size	Agent	22.0	18.0	4.0
	Block	20.0	18.0	16.0
Shape	Agent	26.0	52.0	18.0
	Block	12.0	14.0	8.0

(b) Planning success under factor variations.

Figure 5: Robustness analysis on Push-T. Left: effect of visual distractors. Right: planning success under factor variations.

outside their training distribution. Fig. 11 reports the variations of SR as a function of color wheel background intensities. Additionally, using the occlusion wrapper introduced in Fig. 6, we report in Fig. 5a the planning success rate as a function of the number of visual distractor squares. We observe a quadratic decay: models tolerate small numbers of distractors but degrade rapidly beyond that point. The pattern holds across all baselines. Overall, these results indicate that current world models exhibit limited zero-shot generalization: even modest shifts outside the training distribution lead to substantial degradation in planning performance. `swm` makes such systematic robustness studies trivial to run and reproduce.

These case studies demonstrate how `swm` turns previously ad-hoc analyses into standardized, reproducible experiments, accelerating the identification of current limitations and the development of more robust world models.

5 Related Work

Benchmarking World Models. Learning predictive models of the environment has become a central AI paradigm, driving a steady increase in the development of world models. Generative latent approaches, such as Dreamer [43, 44, 7, 17, 24] rely on pixel reconstruction, while implicit methods like TD-MPC2 [45, 8] optimize for reward prediction. A recent alternative trains purely via self-supervised prediction in learned representation spaces to model dynamics [19, 46, 22, 47, 21, 10]. Despite their shared philosophy, standalone codebases lead to duplicated implementations and inconsistent evaluation, hindering research progress. Recent benchmarks such as WorldMark [48]

and comprehensive frameworks like WorldTest [49] standardize the evaluation of interactive video and behavior-based models, yet their training infrastructure remains fragmented. Conversely, works like EB-JEPA [50] are restricted to JEPA-style architectures, are mainly developed for educational purposes, and lack scalable research components, such as efficient data-loading, training-agnostic evaluation, and modeling baselines. `swm` addresses these shortcomings by providing a model-agnostic, complete ecosystem for efficient training and evaluation.

Benchmarking RL. World model evaluation is closely tied to Reinforcement Learning (RL), where standardized benchmarks have driven substantial progress [51, 52, 43, 53–55]. Datasets such as `D4RL` [56] and `OGBench` [39] dominate offline and goal-conditioned RL evaluations, but focus mainly on model-free approaches using static trajectories in fixed environments. Concurrently, visual robustness has been studied via the `Natural RL Environment` [57], `Distracting Control Suite` [58], `DMControl Generalization Benchmark` [59], and `DMC-VB` [60]. However, these remain tied to specific environment families and only address visual perturbations. Recent efforts have also highlighted the importance of rapid adaptation to structurally novel environments with different dynamic mechanics (e.g., *different games*) [61]. To address this, `swm` reuses environments from these benchmarks and introduces systematic variations to evaluate robustness, zero-shot adaptation, and generalization.

Complementary Frameworks. The open-source ecosystem already supports algorithm development outside of the field of world modeling. For instance, `stable-baselines3` [62] and `CleanRL` [63] standardize model-free baselines, while `mbrl-lib` [64] provides an older, less-maintained model-based counterpart. Robotics frameworks such as `LeRobot` [27], `RoboHive` [65], `robosuite` [66], and `RLBench` [67] focus on imitation learning and policy optimization, leaving world-model infrastructure largely unaddressed. Rather than replacing these ecosystems, `swm` integrates seamlessly with frameworks like `stable-baselines3` and `LeRobot` to supply missing world-model components, like planning or evaluation. `swm` aims to become a unified, standard reference library for the research community.

6 Conclusion and Future Work

We introduced `stable-worldmodel`, an open-source platform for reproducible research and evaluation of world models. `swm` provides efficient, modular, and well-tested tools to reduce the friction in the full world modeling pipeline, including high-performance data management, planning solvers, and standardized evaluation benchmark extended with customisable visual and physical properties. Our experiments highlight that current world models remain far from achieving robust zero-shot generalization, even under mild distribution shifts, underscoring the need for advances in both visual and physical robustness as well as long-horizon consistency. These findings suggest that improved generalization may require both architectural advances and systematic scaling. By enabling a reliable and scalable evaluation workflow, `swm` makes it possible to rigorously study scaling trends and their impact on generalization, supporting more controlled and reproducible investigations of existing and future architectures. Looking forward, we aim to extend the platform beyond simulation toward improved sim-to-real transfer, support for asynchronous real-time interaction, and efficient online training, as well as broaden its applicability to a wider range of scientific domains. We hope this unified framework will foster fairer comparisons, more reliable generalization studies, and accelerated progress toward robust and trustworthy world models for the real world.

References

[1] AI Propoi. Use of linear programming methods for synthesizing sampled-data automatic systems. *Automn. Remote Control*, 24(7):837–844, 1963.

[2] Jacques Richalet. Industrial applications of model based predictive control. *Automatica*, 29(5): 1251–1274, 1993.

[3] Basil Kouvaritakis and Mark Cannon. Model predictive control. *Switzerland: Springer International Publishing*, 38(13-56):7, 2016.

[4] James B Rawlings, David Q Mayne, and Moritz M Diehl. Model predictive control: theory, computation, and design. *(No Title)*, 2020.

[5] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3):440, 2018.

[6] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

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

[8] Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.

[9] Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mido Assran, and Nicolas Ballas. V-jepa: Latent video prediction for visual representation learning. 2023.

[10] Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-jepa 2: Self-supervised video models enable understanding, prediction and planning. *arXiv preprint arXiv:2506.09985*, 2025.

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

[12] Frederick P Brooks Jr. *The mythical man-month: essays on software engineering*. Pearson Education, 1995.

[13] Edward Raff. A step toward quantifying independently reproducible machine learning research. *Advances in Neural Information Processing Systems*, 32, 2019.

[14] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[16] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

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

[18] Yann LeCun et al. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62(1):1–62, 2022.

[19] Vlad Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim GJ Rudner, and Yann LeCun. Learning from reward-free offline data: A case for planning with latent dynamics models. *arXiv preprint arXiv:2502.14819*, 2025.

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

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

[22] Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. *arXiv preprint arXiv:2411.04983*, 2024.

[23] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*, volume 133. Springer, 2004.

[24] Jake Bruce, Michael D Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, et al. Genie: Generative interactive environments. In *Forty-first International Conference on Machine Learning*, 2024.

[25] Ziming Liu, Sophia Sanborn, Surya Ganguli, and Andreas Tolias. From kepler to newton: Inductive biases guide learned world models in transformers, 2026. URL https://arxiv.org/abs/2602.06923.

[26] Weston Pace, Chang She, Lei Xu, Will Jones, Albert Lockett, Jun Wang, and Raunak Shah. Lance: Efficient random access in columnar storage through adaptive structural encodings, 2025. URL https://arxiv.org/abs/2504.15247.

[27] Remi Cadene, Simon Alibert, Francesco Capuano, Michel Aractingi, Adil Zouitine, Pepijn Kooijmans, Jade Choghari, Martino Russi, Caroline Pascal, Steven Palma, et al. Lerobot: An open-source library for end-to-end robot learning. In *The Fourteenth International Conference on Learning Representations*, 2026.

[28] Taylor Howell, Nimrod Gileadi, Saran Tunyasuvunakool, Kevin Zakka, Tom Erez, and Yuval Tassa. Predictive sampling: Real-time behaviour synthesis with mujoco. 2022.

[29] Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient cross-entropy method for real-time planning. In *Conference on Robot Learning*, pages 1049–1065. PMLR, 2021.

[30] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1433–1440. IEEE, 2016.

[31] Mikael Henaff, William F Whitney, and Yann LeCun. Model-based planning with discrete and continuous actions. *arXiv preprint arXiv:1705.07177*, 2017.

[32] Michael Psenka, Michael Rabbat, Aditi Krishnapriyan, Yann LeCun, and Amir Bar. Parallel stochastic gradient-based planning for world models. *arXiv preprint arXiv:2602.00475*, 2026.

[33] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning, 2020. URL https://arxiv.org/abs/1912.06088.

[34] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[35] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[36] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[37] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 2012.

[38] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

[39] Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *The Thirteenth International Conference on Learning Representations*, 2025.

[40] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of artificial intelligence research*, 47:253–279, 2013.

[41] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[42] Michael Matthews, Michael Beukman, Benjamin Ellis, Mikayel Samvelyan, Matthew Jackson, Samuel Coward, and Jakob Foerster. Craftax: A lightning-fast benchmark for open-ended reinforcement learning. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pages 35104–35137, 2024. URL https://arxiv.org/abs/2402.16801.

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

[44] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[45] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*, 2022.

[46] Quentin Garrido, Mahmoud Assran, Nicolas Ballas, Adrien Bardes, Laurent Najman, and Yann LeCun. Learning and leveraging world models in visual representation learning. *arXiv preprint arXiv:2403.00504*, 2024.

[47] Amir Bar, Gaoyue Zhou, Danny Tran, Trevor Darrell, and Yann LeCun. Navigation world models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 15791–15801, 2025.

[48] Xiaojie Xu, Zhengyuan Lin, Kang He, Yukang Feng, Xiaofeng Mao, Yuanyang Yin, Kaipeng Zhang, and Yongtao Ge. Worldmark: A unified benchmark suite for interactive video world models. *arXiv preprint arXiv:2604.21686*, 2026.

[49] Archana Warrier, Dat Nguyen, Michelangelo Naim, Moksh Jain, Yichao Liang, Karen Schroeder, Cambridge Yang, Joshua B Tenenbaum, Sebastian Vollmer, Kevin Ellis, et al. Benchmarking world-model learning. *arXiv preprint arXiv:2510.19788*, 2025.

[50] Basile Terver, Randall Balestriero, Megi Dervishi, David Fan, Quentin Garrido, Tushar Nagarajan, Koustuv Sinha, Wancong Zhang, Mike Rabbat, Yann LeCun, et al. A lightweight library for energy-based joint-embedding predictive architectures. *arXiv preprint arXiv:2602.03604*, 2026.

[51] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[52] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[53] 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*, pages 1861–1870. Pmlr, 2018.

[54] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[55] Eloi Alonso, Adam Jelley, Vincent Micheli, Anssi Kanervisto, Amos Storkey, Tim Pearce, and François Fleuret. Diffusion for world modeling: Visual details matter in atari. *Advances in Neural Information Processing Systems*, 37:58757–58791, 2024.

[56] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[57] Amy Zhang, Yuxin Wu, and Joelle Pineau. Natural environment benchmarks for reinforcement learning. *arXiv preprint arXiv:1811.06032*, 2018.

[58] Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite–a challenging benchmark for reinforcement learning from pixels. *arXiv preprint arXiv:2101.02722*, 2021.

[59] Nicklas Hansen, Hao Su, and Xiaolong Wang. Stabilizing deep q-learning with convnets and vision transformers under data augmentation. *Advances in neural information processing systems*, 34:3680–3693, 2021.

[60] Joseph Ortiz, Antoine Dedieu, Wolfgang Lehrach, J Swaroop Guntupalli, Carter Wendelken, Ahmad Humayun, Sivaramakrishnan Swaminathan, Guangyao Zhou, Miguel Lázaro-Gredilla, and Kevin P Murphy. Dmc-vb: A benchmark for representation learning for control with visual distractors. *Advances in Neural Information Processing Systems*, 37:6574–6602, 2024.

[61] Lance Ying, Katherine M Collins, Prafull Sharma, Cedric Colas, Kaiya Ivy Zhao, Adrian Weller, Zenna Tavares, Phillip Isola, Samuel J Gershman, Jacob D Andreas, et al. Assessing adaptive world models in machines with novel games. *arXiv preprint arXiv:2507.12821*, 2025.

[62] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of machine learning research*, 22(268):1–8, 2021.

[63] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and JoÃǪo GM AraÃšjo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

[64] Luis Pineda, Brandon Amos, Amy Zhang, Nathan O Lambert, and Roberto Calandra. Mbrl-lib: A modular library for model-based reinforcement learning. *arXiv preprint arXiv:2104.10159*, 2021.

[65] Vikash Kumar, Rutav Shah, Gaoyue Zhou, Vincent Moens, Vittorio Caggiano, Abhishek Gupta, and Aravind Rajeswaran. Robohive: A unified framework for robot learning. *Advances in Neural Information Processing Systems*, 36:44323–44340, 2023.

[66] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Kevin Lin, Abhiram Maddukuri, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

[67] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2): 3019–3026, 2020.

[68] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*, 2019.

[69] Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. 2021.

[70] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 272–279, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390191. URL https://doi.org/10.1145/1390156.1390191.

[71] Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019.

[72] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022.

Appendix

Our Appendix complements the main paper with a walkthrough of the `stable-worldmodel` platform. We begin with installation and environment setup, before turning to the core abstractions of the library: `World`, `Policy`, `Solver`, factors of variation, wrappers, and the dataset API. Building on these foundations, we detail the supported environment families along with the two complementary intervention mechanisms used to construct distribution shifts, namely the factors of variation and the wrappers. We then review the implemented baselines, presenting their training objectives and accompanying pseudo-code, followed by the evaluation protocols supported by the platform. Next, we cover the supported planning solvers, spanning both sampling-based and gradient-based methods, each illustrated, again, with their own pseudocode. We also document the ready-to-use scripts and command-line interface that make the platform immediately usable in practice. Finally, we outline open research problems that the platform makes tractable, provide concrete guidelines for contributors adding new environments, solvers, or baselines, and close with additional experimental results.

A Installation & Setup

This section walks through the process of installing `stable-worldmodel` and preparing your environment for development and experimentation. We recommend using `uv` as your Python package manager and Linux as the operating system for the best experience. The library should also work on macOS or Windows, and an alternative setup using Conda is possible.

`swm` is designed to be easy to set up while supporting both local development and large-scale distributed training.

Development setup:

```
[assuming downloaded code]
cd stable-worldmodel
uv venv --python=3.10 && source .venv/bin/activate
uv sync --all-extras --group dev
```

Note: `swm` uses the environment variable `$STABLEWM_HOME` (default: `~/.stable_worldmodel`) to locate datasets and checkpoints. Datasets are stored in `$STABLEWM_HOME/datasets` and checkpoints in `$STABLEWM_HOME/checkpoints`.

B Core Abstractions

The design of `stable-worldmodel` revolves around a small set of clean, composable abstractions that unify data collection, training, and evaluation of world models. These abstractions are deliberately lightweight, compatible with the PyTorch deep learning framework and Gymnasium-based environments, while adding powerful features tailored for world model research.

The central abstraction is the `World` class. A `World` wraps one or more vectorized Gymnasium environments through an efficient internal `EnvPool` and provides a unified interface for simulation, dataset recording, visualization, and evaluation. Unlike the standard Gymnasium API, `World.reset()` and `World.step()` do not return observations or rewards. Instead, they update an internal dictionary called `world.infos` in place. This dictionary contains all information returned by the underlying environments (RGB frames, states, rewards, terminations, truncations, etc.), giving immediate and consistent access to the complete state of all environments at every timestep. Note that each call to `reset()` or `step()` replaces the previous `world.infos` content.

Action selection is decoupled via a `Policy` object. After calling `world.set_policy(policy)`, every call to `World.step()` automatically queries the associated policy for actions. This design makes it trivial to swap between expert policies, random policies, learned policies, or planning-based controllers at any time without modifying the world loop. A policy only needs to implement a `get_actions(info)` method that receives the current `infos` dictionary. `swm` provides several ready-to-use policies, including heuristic policies, `FeedForwardPolicy` for reinforcement learning agents, and `MPCPolicy` for test-time planning.

Planning and model-predictive control are supported via `Solver` implementations (e.g., `CEMSolver`, `MPPISolver`) and the `MPCPolicy` abstraction. Any world model that implements a `get_cost` method can be directly turned into a deployable controller.

Another key abstraction is the notion of **Factors of Variation (FoV)**. Every supported environment exposes a set of controllable properties (colors, shapes, sizes, physical parameters, lighting, etc.) through a dedicated `variation_space`. These factors can be fixed, randomly sampled, or systematically varied during data collection or evaluation, enabling rigorous studies of robustness, domain shift, continual learning, and zero-shot generalization.

The `World` interface also supports `Wrapper` objects that can be easily attached to modify parts of the `infos` dictionary or alter default behavior (e.g., adding observations, changing rewards, applying visual transformations, or injecting custom logic). For more details on wrappers, see Appendix C or Section 3 of the paper.

Data handling is managed through the `Dataset` API. `swm` supports multiple storage formats: Lance, HDF5, MP4 (video), and LeRobot, through a flexible format registry. Each format provides its own reader and writer implementation, making it straightforward to swap formats or add new ones. A convenient `swm.data.convert()` utility allows one-line conversion between any supported formats (provided the corresponding reader and writer exist).

Together, these abstractions create a highly modular pipeline where researchers can focus on their algorithm evaluation while relying on `swm` for reliable environments, efficient data pipelines, and solver algorithms. For a more detailed description of each abstraction, we refer the reader to the documentation.

C Environments and Factors of Variation (FoV)

The main strength of `swm` lies in its large, standardized, and highly customizable suite of environments. All environments follow the standard `Gymnasium` interface while being deeply integrated with the `World` abstraction, enabling high-throughput data collection, visualization, and evaluation as mentioned earlier.

`swm` supports environments from a wide range of domains:

- **DeepMind Control Suite (DMC)**: locomotion, control and manipulation tasks (continuous)
- **OGBench**: 3D object manipulation and scene understanding (continuous)
- **Classic Control Suite**: Classic control tasks (mountain car, inverted pendulum, etc.) (continuous/discrete)
- **Fetch-Suite**: 3D robotic manipulation tasks (continuous)
- **Craftax**: open-world procedural 2D survival game (pixel & symbolic) (discrete)
- **Arcade Learning Environment (ALE)**: 100+ Atari games (discrete)
- **Extra Environments**: `swm/PushT-v1` (2D manipulation) and `swm/TwoRoom-v1` (simple navigation) (continuous)

A `World` can be instantiated with any registered environment ID. For example:

```
import stable_worldmodel as swm
world = swm.World("swm/PushT-v1", num_envs=8, image_shape=(64, 64))
```

Note: `swm` can load by default any environment registered through Gymnasium and work with them. However, unless explicitly added, no Factors of Variation will be available for these non-supported environments.

C.1 Environment Registry

Every environment supported by `swm` is registered under a Gymnasium-compatible ID and can be instantiated directly through the `World` interface. The registry distinguishes two tiers of environments based on the level of integration with `swm`'s intervention machinery.

Native environments (registered under the `swm/*` namespace) expose a structured state dictionary in their `info` output and declare a typed `variation_space` listing all available factors of variation (FoV). This allows simulator-level interventions: scene properties, geometry, and physical parameters can be edited at reset time through the standard Gymnasium `options` argument.

External environments (e.g., ALE, Craftax) are imported as-is and run through `World` without modification. They do not natively expose a `variation_space`; interventions on these environments are instead applied at the observation boundary through `Visual Wrappers` (Sec. C.2), or via a dedicated adapter when one is provided.

C.2 Factors of Variation (FoV) & Wrappers

`swm` provides two parallel mechanisms for these interventions, described next: a simulator-level mechanism for environments whose internal state we can edit, and a boundary-level mechanism based on Gymnasium wrappers for environments we treat as black boxes (Atari, Craftax). Both feed into the same evaluation pipeline and can be active on a single `World` instance.

Factor of Variation. Every native `swm` environment exposes its interventions through a hierarchical `variation_space`. The hierarchy follows the structure of the scene: Push-T exposes factors for the agent, block, goal, and background, while MuJoCo-based environments expose factors tied to bodies, geoms, joints, cameras, and physics parameters. FoVs are configured through the standard Gymnasium `options` argument, which is forwarded by `World.reset()`, `World.collect()`, `World.evaluate()`, `evaluate_from_dataset()`, and related methods. Factors are referenced by dot-key names such as `agent.color`, `block.scale`, or `physics.floor.friction`, and setting `variation` to `"all"` samples every available native factor.

FoVs can be sampled randomly or pinned to fixed values. Random sampling produces train-time diversity or evaluation-time distribution shifts; fixed values give controlled ablations, reproducible sweeps, and debugging targets. In both cases the sampled values are applied at reset and held constant for the duration of the episode. That matters for evaluation: a failure can be attributed to a persistent change in the environment, not to frame-wise noise injected independently at every timestep.

```
# Example when collecting a dataset
world.collect(
    "data/pusht_demo.h5",
    episodes=100,
    options={
        "variation": ["agent.color", "block.scale"],
        "variation_values": {
            "background.color": [0.1, 0.2, 0.9] # fix specific value
        }
    }
)

# Example during evaluation
world.evaluate(
    policy=your_policy,
    options={"variation": ["all"]}
)
```

Boundary-level FoV via Visual Wrappers. Not all environments expose editable simulator internals. Atari games run from ROMs, and Craftax exposes only the observation pipeline without programmatic hooks into rendering or transitions. `swm` handles these cases through Gymnasium wrappers, used as boundary-level interventions: a wrapper sits between the environment and the agent and overwrites the relevant channel (frames, rewards, actions, or auxiliary metadata) without touching the original source. We focus here on visual wrappers, listed in Table 2: the environment renders its RGB frame, and the wrapper rewrites that frame before it reaches the agent, policy, or world model. Figure 6 shows the resulting perturbations on a Craftax frame.

Table 2: Visual wrappers available in `swm`. These wrappers operate on rendered frames and can therefore be applied even when simulator internals are inaccessible.

Wrapper	Intervention
`ChromaKeyWrapper`	Replaces pixels matching a target color with a user-provided image or video background.
`NoiseWrapper`	Adds Gaussian or salt-and-pepper pixel noise.
`BlurWrapper`	Applies Gaussian blur with a controllable kernel size.
`ColorJitterWrapper`	Perturbs brightness, contrast, saturation, and hue.
`GrayscaleWrapper`	Removes color information while preserving spatial structure.
`RandomShiftWrapper`	Translates the frame by a random offset.
`CutoutWrapper`	Masks random rectangular regions of the frame.
`OcclusionWrapper`	Places solid occluding patches at random locations.
`MovingPatchWrapper`	Adds occluding patches with independent temporal motion.
`RandomConvWrapper`	Applies a fixed random convolution to the frame.
`ResolutionWrapper`	Downsamples and upsamples frames to vary spatial fidelity.

Wrappers are passed to `World` through the `extra_wrappers` argument as a list of factories, applied in order on top of `swm`'s default `MegaWrapper`. The same call site instruments a native MuJoCo task and a closed-source ROM; only the contents of the list change.

```
from functools import partial
import stable_worldmodel as swm
from stable_worldmodel.wrapper.visual import (
    NoiseWrapper, ColorJitterWrapper, OcclusionWrapper,
)

# Robustness sweep on Atari Pong: pixel noise, color jitter, occluders.
world = swm.World(
    "ALE/Pong-v5",
    num_envs=8,
    image_shape=(84, 84),
    extra_wrappers=[
        partial(NoiseWrapper, std=8.0),
        partial(ColorJitterWrapper, brightness=0.3, hue=0.1),
        partial(OcclusionWrapper, num_patches=2, size=(0.1, 0.25)),
    ],
)
```

Each wrapper takes its own constructor arguments (kernel size, noise standard deviation, occlusion fraction, and so on). Wrappers that resample per episode, such as `ColorJitterWrapper`, `OcclusionWrapper`, and `RandomShiftWrapper`, redraw their internal parameters at every `reset`, so a sweep produces a fresh draw per episode rather than a single fixed perturbation. The two FoV mechanisms compose: a single `World` can stack visual wrappers over a native FoV environment by passing `extra_wrappers` at construction together with an `options={"variation": [...]}` dictionary to `World.reset()`.

Note: `swm` can load by default any environment registered through Gymnasium and work with them. However, unless explicitly added, no Factors of Variation will be available for these non-supported environments. Fig. 7 summarizes the number of native and wrapper-based FoV exposed by each supported environment.

D Available Baselines

`swm` provides six implementations of world model baselines. These cover two major paradigms for solving goal-conditioned environments: goal-conditioned reinforcement learning (GCRL) and latent world models. All baselines share a common structure, making it simple to create direct comparisons. For training, all baselines share a common Hydra-configured training entry point and matching config. All algorithms described learn offline from a dataset D consisting of trajectories $(o_0, a_0, o_1, \ldots, o_T)$ where o_i and a_i are the observation and action at time i respectively. These

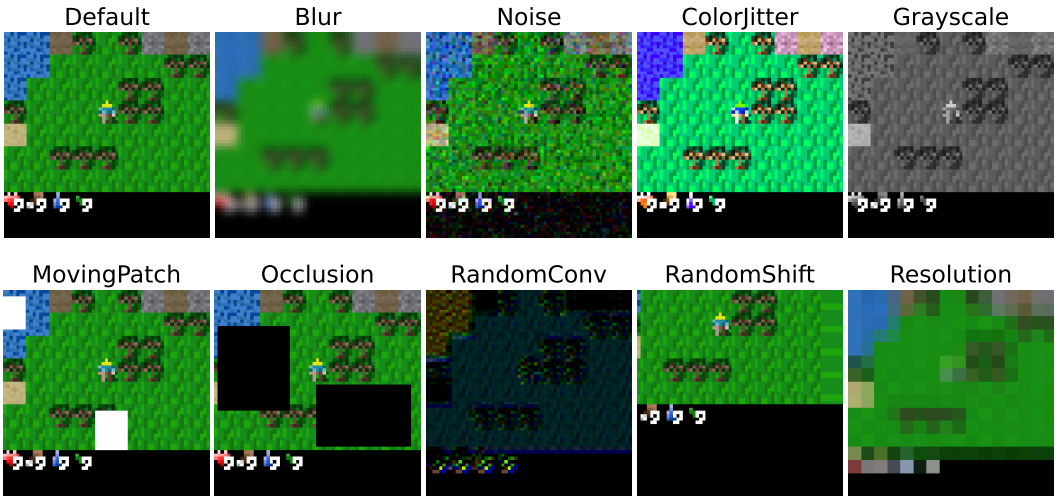


Figure 6: Examples of visual perturbations that can be applied on-the-fly to any supported environment without modifying its source code. These perturbations enable systematic robustness and generalization studies even for environments with limited source access (e.g., Atari ROMs).

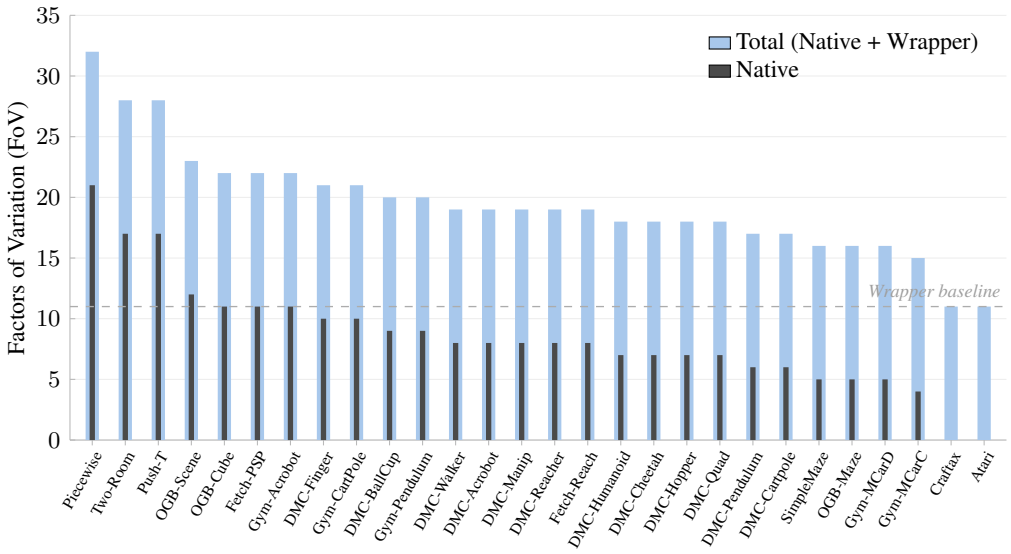


Figure 7: Factors of variation supported per environment. Light blue bars indicate total factors available (native + 11 universal visual wrappers); inset dark bars show only those exposed natively. Closed-source simulators (Atari, Craftax) lack accessible internals and rely entirely on the wrapper layer; Atari denotes the ALE suite (100+ games), which contributes no native simulator FoV here.

data are collected before training and can be shared across baselines, allowing fair comparisons of offline methods. As described earlier, checkpoints live in `$ STABLEWM_HOME/checkpoints`. GCRL baselines implement `Actionable` and are wrapped by a common `FeedForwardPolicy` with no solver. Latent world models implement `Costable` and are wrapped by `MPCPolicy`, along with any solver described in Appendix F.

D.1 Goal-Conditioned Reinforcement Learning

Goal-conditioned reinforcement learning follows a similar structure to classical reinforcement learning, with learned functions having an extra parameter of the current goal. The GCRL baselines implemented by `swm` share a common `GCRL` class which combines a DINOv2 head with, depending on the algorithm, an action predictor, a value predictor, and a critic predictor. While not in the

original implementations, in `swm` observations and goals are first encoded into DINOv2 [35] patch embeddings before being used as states. All GCRL baselines act through `get_action` which takes a current state and goal and returns an action. GCRL baselines do not use any predictors and, as such, do not use any solvers.

Goal-conditioned Behavioural Cloning Goal-Conditioned Behavioural Cloning (GCBC) [68] is an imitation learning baseline that learns a goal-conditioned policy $\pi(a|s, g)$. It is trained to reproduce expert actions given a state and goal.

Algorithm 2: GCBC Training

Input: Dataset D of expert trajectories, frozen encoder $\mathcal{E}$ (DINOv2), goal-conditioned policy π_θ, history H, learning rate η.
Output: Policy π_θ.
while *not converged* **do**
  Sample $(o_0, \ldots, o_{H-1},\ a_0, \ldots, a_{H-1},\ g) \sim D$
  for $i \leftarrow 0$ **to** $H-1$ **do**
    $z_i \leftarrow \mathcal{E}(o_i)$
  end
  $z_g \leftarrow \mathcal{E}(g)$
  for $i \leftarrow 0$ **to** $H-1$ **do**
    $\hat{a}_i \leftarrow \pi_\theta(z_i,\ z_g)$
  end
  $\mathcal{L} \leftarrow \dfrac{1}{H} \displaystyle\sum_{i=0}^{H-1} \|\hat{a}_i - a_i\|_2^2$
  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$
end

Implicit Q-Learning In implicit q-learning (IQL), a state value function is learned through expectile regression. This is done through jointly learning a critic and value function. After these networks are trained, a policy is extracted via advantage-weighted regression (AWR). `swm` implements two variations: implicit q-learning and implicit v-learning. IQL learns both a Q-function $Q_\psi(s_t, a_t, g)$ and a value function $V_\theta(s_t, g)$. The Q-network is trained with Bellman regression, bootstrapping from the target value network $V_{\bar{\theta}}$:

$$\mathcal{L}_Q = \mathbb{E}_{(s_t, a_t, s_{t+1}, g) \sim \mathcal{D}} \left[\left(Q_\psi(s_t, a_t, g) - \left(r(s_t, g) + \gamma\, m_t\, V_{\bar{\theta}}(s_{t+1}, g) \right) \right)^2 \right]$$

where $m_t = 0$ if $s_t = g$ (terminal) and $m_t = 1$ otherwise. The value network is trained with expectile regression against targets from the target Q-network $Q_{\bar{\psi}}$:

$$\mathcal{L}_V = \mathbb{E}_{(s_t, a_t, g) \sim \mathcal{D}} \left[L_\tau^2 \big(Q_{\bar{\psi}}(s_t, a_t, g) - V_\theta(s_t, g) \big) \right]$$

where $L_\tau^2(u) = |\tau - \mathbb{1}(u < 0)|\, u^2$ is the asymmetric expectile loss. The total critic-phase loss is $\mathcal{L}_{\text{critic}} = \mathcal{L}_Q + \mathcal{L}_V$.

Algorithm 3: GCIQL Training

Input: Dataset D, frozen encoder $\mathcal{E}$ (DINOv2), value head V_θ with target $V_{\bar{\theta}}$, critic Q_ψ with target $Q_{\bar{\psi}}$, actor π_ϕ, discount γ, expectile τ, AWR temperature α, EMA rate ρ, learning rate η.
Output: Value V_θ, critic Q_ψ, actor π_ϕ

`// Phase 1: joint value and critic learning`
while not converged **do**

Sample $(o_t, a_t, o_{t+1}, g) \sim D$
$z_t, z_{t+1}, z_g \leftarrow \mathcal{E}(o_t), \mathcal{E}(o_{t+1}), \mathcal{E}(g)$
$m \leftarrow 1 - \mathbb{1}[o_t = g]$
$r \leftarrow -m$
$q_{\text{tgt}} \leftarrow r + \gamma\, m\, V_{\bar{\theta}}(z_{t+1}, z_g)$
$\mathcal{L}_Q \leftarrow \big(Q_\psi(z_t, a_t, z_g) - q_{\text{tgt}}\big)^2$
$\mathcal{L}_V \leftarrow L_\tau^2\big(Q_{\bar{\psi}}(z_t, a_t, z_g) - V_\theta(z_t, z_g)\big)$
$\mathcal{L}_{\text{critic}} \leftarrow \mathcal{L}_Q + \mathcal{L}_V$
$(\theta, \psi) \leftarrow (\theta, \psi) - \eta\, \nabla \mathcal{L}_{\text{critic}}$
$(\bar{\theta}, \bar{\psi}) \leftarrow \rho\, (\bar{\theta}, \bar{\psi}) + (1 - \rho)\, (\theta, \psi)$

// Phase 2: policy extraction with V_θ, Q_ψ frozen
while not converged **do**
Sample $(o_t, a_t, g) \sim D$
$z_t, z_g \leftarrow \mathcal{E}(o_t), \mathcal{E}(g)$
$A \leftarrow Q_\psi(z_t, a_t, z_g) - V_\theta(z_t, z_g)$
$w \leftarrow \exp(\alpha\, A)$
$\mathcal{L}_\pi \leftarrow w \cdot \|\pi_\phi(z_t, z_g) - a_t\|_2^2$
$\phi \leftarrow \phi - \eta\, \nabla_\phi \mathcal{L}_\pi$

Implicit V-Learning Implicit v-learning is an extension of IQL that learns only a value function and no q-function. It does this by training $V_\theta(s_t, g)$ directly against bootstrapped targets from a target network $V_{\bar{\theta}}$:

$$\mathcal{L}_V = \mathbb{E}_{(s_t, s_{t+1}, g) \sim \mathcal{D}} \left[L_\tau^2(r(s_t, g) + \gamma\, V_{\bar{\theta}}(s_{t+1}, g) - V_\theta(s_t, g)) \right]$$

with the same expectile loss L_τ^2, discount γ, and reward r as IQL.

Algorithm 4: GCIVL Training

Input: Dataset D, frozen encoder $\mathcal{E}$ (DINOv2), value head V_θ with target $V_{\bar{\theta}}$, actor π_ϕ, discount γ, expectile τ, AWR temperature α, EMA rate ρ, learning rate η
Output: Value V_θ, critic Q_ψ, actor π_ϕ.

// Phase 1: joint value and critic learning
while not converged **do**
Sample $(o_t, a_t, o_{t+1}, g) \sim D$
$z_t, z_{t+1}, z_g \leftarrow \mathcal{E}(o_t), \mathcal{E}(o_{t+1}), \mathcal{E}(g)$
$m \leftarrow 1 - \mathbb{1}[o_t = g]$
$r \leftarrow -m$
$q \leftarrow r + \gamma\, m\, V_{\bar{\theta}}(z_{t+1}, z_g)$
$\mathcal{L}_V \leftarrow L_\tau^2\big(q - V_\theta(z_t, z_g)\big)$
$\theta \leftarrow \theta - \eta\, \nabla_\theta \mathcal{L}_V$
$\bar{\theta} \leftarrow \rho\, \bar{\theta} + (1 - \rho)\, \theta$

// Phase 2: policy extraction with V_θ frozen
while not converged **do**
Sample $(o_t, a_t, g) \sim D$
$z_t, z_g \leftarrow \mathcal{E}(o_t), \mathcal{E}(g)$
$A \leftarrow V_\theta(z_{t+1}, z_g) - V_\theta(z_t, z_g)$
$w \leftarrow \exp(\alpha\, A)$
$\mathcal{L}_\pi \leftarrow w \cdot \|\pi_\phi(z_t, z_g) - a_t\|_2^2$
$\phi \leftarrow \phi - \eta\, \nabla_\phi \mathcal{L}_\pi$

D.2 Latent World Models

Latent world models consist of an encoded $\mathcal{E}$ that maps observations to a latent space and a predictor $\mathcal{P}$ that models the latent dynamics. The encoder is a function of observations, and produces an embedding. The predictor is a function of a history of H embeddings, concatenated with action embeddings, and predicts the embedding k time steps into the future, where k is the offset. At test time, all baselines are wrapped by a common `MPCPolicy` that includes a solver Appendix F. Each world model must implement `encode`, `predict`, `rollout`, `criterion`, and `get_cost`. The solver uses these functions to evaluate candidate action sequences by rolling out the predictor and scoring the resulting trajectories, then returns the best found action sequence. This process is described in more depth in Appendix F.

DINO-WM DINO-WM [22] is a latent world model with a frozen, pretrained DINOv2 [35] encoder. Only the predictor is learned, operating over patch-level visual tokens produced by the encoder, as well as optionally proproiception embeddings. The predictor is a causal Transformer with a windowed attention mask of size H. It is trained by teacher-forced ℓ_2-regression toward the DINOv2 embedding of the next frame. By freezing the encoder, DINOv2 avoids collapse common in JEPAs without regularization. DINO-WM serves as proof of latent planning using visual features, even without end-to-end encoder-predictor training.

Algorithm 5: DINO-WM Training

Input: Dataset D, frozen encoder $\mathcal{E}$ (DINOv2), predictor $\mathcal{P}_\theta$, action encoder $\mathcal{A}$; history H, offset k, learning rate η
Output: Predictor $\mathcal{P}_\theta$.
while not converged **do**
  Sample $(o_0, \ldots, o_{H+k-1},\ a_0, \ldots, a_{H-1}) \sim D$
  for $i \leftarrow 0$ **to** $H + k - 1$ **do**
    $z_i \leftarrow \mathcal{E}(o_i)$
  for $i \leftarrow 0$ **to** $H - 1$ **do**
    $e_i \leftarrow \mathrm{concat}(z_i,\ \mathcal{A}(a_i))$
  $(\hat{z}_k, \ldots, \hat{z}_{H+k-1}) \leftarrow \mathcal{P}_\theta(e_0, \ldots, e_{H-1})$
  $\mathcal{L} \leftarrow \dfrac{1}{H} \displaystyle\sum_{i=0}^{H-1} \left\| \hat{z}_{i+k} - z_{i+k} \right\|_2^2$
  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}$

Planning with Latent Dynamics Models Planning with Latent Dynamics Models (PLDM) [19] is an end-to-end JEPA. The encoder and predictor are trained jointly from raw observations. The encoder produces a single `[CLS]`-token embedding, as opposed to full patch-level visual tokens, that is used by the predictor. To prevent representational collapse, PLDM uses a multi-term objective, combining teacher-forced prediction loss with VICReg-style variance and covariance regularizers [69], a temporal alignment term, and an inverse dynamics term, each with their own independent weight.

Algorithm 6: PLDM Training

Input: Dataset D, encoder $\mathcal{E}_\theta$, predictor $\mathcal{P}_\theta$, action encoder $\mathcal{A}_\theta$, inverse dynamics model IDM_ψ, history H, loss weights $\alpha, \beta, \delta, \omega$, learning rate η
Output: Encoder $\mathcal{E}_\theta$ and predictor $\mathcal{P}_\theta$.
while not converged **do**

Sample $(o_0, \ldots, o_T,\ a_0, \ldots, a_{T-1}) \sim D$
for $i \leftarrow 0$ **to** T **do**
  $z_i \leftarrow \mathcal{E}_\theta(o_i)$
$(\hat{z}_1, \ldots, \hat{z}_T) \leftarrow \mathcal{P}_\theta\big(z_{0:T-1},\ \mathcal{A}_\theta(a_{0:T-1})\big)$

$$\mathcal{L}_{\text{sim}} \leftarrow \frac{1}{T}\sum_{i=1}^{T} \|\hat{z}_i - z_i\|_2^2$$

$$\mathcal{L}_{\text{std}} \leftarrow \text{VarLoss}(z_{0:T})$$

$$\mathcal{L}_{\text{cov}} \leftarrow \text{CovLoss}(z_{0:T})$$

$$\mathcal{L}_{\text{temp}} \leftarrow \frac{1}{T}\sum_{i=0}^{T-1} \|z_i - z_{i+1}\|_2^2$$

for $i \leftarrow 0$ **to** $T-1$ **do**
  $\hat{a}_i \leftarrow \text{IDM}_\psi(z_i,\ z_{i+1})$

$$\mathcal{L}_{\text{idm}} \leftarrow \frac{1}{T}\sum_{i=0}^{T-1} \|\hat{a}_i - a_i\|_2^2$$

$$\mathcal{L} \leftarrow \mathcal{L}_{\text{sim}} + \alpha\,\mathcal{L}_{\text{std}} + \beta\,\mathcal{L}_{\text{cov}} + \delta\,\mathcal{L}_{\text{temp}} + \omega\,\mathcal{L}_{\text{idm}}$$
$$(\theta, \psi) \leftarrow (\theta, \psi) - \eta\,\nabla\mathcal{L}$$

LeWorldModel LeWorldModel (LeWM) [21] follows the same architecture as PLDM. It is an end-to-end JEPA that jointly trains its encoder and predictor. Instead of the five-term anti-collapse objective, LeWM uses a single regularizer: SIGReg, which pushes the distribution of the latent embeddings to an isotropic Gaussian. As a result, in practice, it only has one effective hyperparameter that requires tuning: the weight of regularization.

Algorithm 7: LeWM Training

Input: Dataset D, encoder $\mathcal{E}_\theta$, predictor $\mathcal{P}_\theta$, action encoder $\mathcal{A}_\theta$, SIGReg weight λ, learning rate η
Output: Encoder $\mathcal{E}_\theta$ and predictor $\mathcal{P}_\theta$.
while not converged **do**
  Sample $(o_0, \ldots, o_T,\ a_0, \ldots, a_{T-1}) \sim D$
  for $i \leftarrow 0$ **to** T **do**
    $z_i \leftarrow \mathcal{E}_\theta(o_i)$
  $(\hat{z}_1, \ldots, \hat{z}_T) \leftarrow \mathcal{P}_\theta\big(z_{0:T-1},\ \mathcal{A}_\theta(a_{0:T-1})\big)$

$$\mathcal{L}_{\text{pred}} \leftarrow \frac{1}{T}\sum_{i=1}^{T} \|\hat{z}_i - z_i\|_2^2$$

$$\mathcal{L}_{\text{SIGReg}} \leftarrow \text{SIGReg}(z_{0:T})$$

$$\mathcal{L} \leftarrow \mathcal{L}_{\text{pred}} + \lambda\,\mathcal{L}_{\text{SIGReg}}$$
$$\theta \leftarrow \theta - \eta\,\nabla_\theta\mathcal{L}$$

TD-MPC2 TD-MPC2 [8] is a reward-driven implicit world model. It consists of an encoder, latent dynamics model, reward predictor, Q function, and stochastic policy prior, all jointly trained. At test time, actions are selected by sample-based planning that bootstraps with the learned terminal value. The model is trained to match predicted next latents with the (stop-gradded) encodings of the corresponding next observations, as done in other latent world models. Additionally, the model is trained to minimize a cross-entropy loss for reward and value prediction and a maximum-entropy policy loss.

Algorithm 8: TD-MPC2 Training

Input: Dataset D of trajectories with observations, actions, and rewards; encoder h, dynamics d, reward R, Q-ensemble $\{Q_i\}_{i=1}^{N_Q}$, target ensemble $\{\bar{Q}_i\}_{i=1}^{N_Q}$, policy prior p (parameters θ); horizon H,

temporal weight ρ, discount γ; loss weights $\beta_c, \beta_r, \beta_v$; entropy coefficient β_π; EMA rate τ; running scale S; learning rate η.
Output: Trained world model θ (encoder, dynamics, reward, Q-ensemble, policy prior).

while not converged **do**
  Sample $(o_0, \ldots, o_H,\ a_0, \ldots, a_{H-1},\ r_0, \ldots, r_{H-1}) \sim D$
  for $i \leftarrow 0$ **to** H **do**
    $\tilde{z}_i \leftarrow h(o_i)$
  $z \leftarrow \tilde{z}_0$
  $\mathcal{L}_\mathrm{c}, \mathcal{L}_\mathrm{r}, \mathcal{L}_\mathrm{v}, \mathcal{L}_\pi \leftarrow 0$
  for $t \leftarrow 0$ **to** $H-1$ **do**
    $z'_t \leftarrow d(z, a_t)$
    $\hat{r}_t \leftarrow R(z, a_t)$
    $\mathcal{L}_\mathrm{c} \mathrel{+}= \rho^t \, \|z'_t - \mathrm{sg}(\tilde{z}_{t+1})\|_2^2$
    $\mathcal{L}_\mathrm{r} \mathrel{+}= \rho^t \, \mathrm{CE}(\hat{r}_t,\ \mathrm{TwoHot}(r_t))$
    begin
      $a'_t \sim p(\cdot \mid \tilde{z}_{t+1})$
      $\{i, j\} \subset \{1, \ldots, N_Q\}$
      $q^\star \leftarrow \min\big(\bar{Q}_i(\tilde{z}_{t+1}, a'_t),\ \bar{Q}_j(\tilde{z}_{t+1}, a'_t)\big)$
      $y_t \leftarrow r_t + \gamma \, q^\star$
    end
    for $i \leftarrow 1$ **to** N_Q **do**
      $\mathcal{L}_\mathrm{v} \mathrel{+}= \rho^t \, \mathrm{CE}(Q_i(z, a_t),\ \mathrm{TwoHot}(y_t))$
    $a^\pi_t \sim p(\cdot \mid \mathrm{sg}(z))$
    $\mathcal{H}_t \leftarrow$ closed-form entropy of p
    $\bar{q}^\pi_t \leftarrow \frac{1}{2}(Q_{i'}(z, a^\pi_t) + Q_{j'}(z, a^\pi_t))$ for random $\{i', j'\}$
    if $t = 0$ **then**
      Update S from quantiles of $\bar{q}^\pi_t$
    $\mathcal{L}_\pi \mathrel{+}= \rho^t \left(- \bar{q}^\pi_t / S - \beta_\pi |\mathcal{A}| \, \mathcal{H}_t \right)$
    $z \leftarrow z'_t$
  $\mathcal{L} \leftarrow \beta_c \, \mathcal{L}_\mathrm{c} + \beta_r \, \mathcal{L}_\mathrm{r} + \beta_v \, \mathcal{L}_\mathrm{v} / N_Q + \mathcal{L}_\pi$
  Update $\theta \leftarrow \theta - \eta \, \nabla_\theta \mathcal{L}$
  for $i \leftarrow 1$ **to** N_Q **do**
    $\bar{Q}_i \leftarrow (1 - \tau) \, \bar{Q}_i + \tau \, Q_i$

E Evaluation Protocols

Goal-observation tasks. `swm` evaluates goal-conditioned agents by specifying tasks through observations rather than through environment-specific reward code. The agent receives the current observation together with a target frame representing the desired state, and is judged on whether its actions bring the environment to that goal from a given initial condition. This convention is shared by planners built on latent world models and by goal-conditioned policies: the policy interface differs between methods, but the evaluation question is the same. Given a start observation and a goal observation, the agent must reach the goal within a fixed interaction budget.

Episodic evaluation. `world.evaluate(episodes=`N`, seed=`s`)` runs the standard online protocol. The environment is reset through the usual Gymnasium reset path, the task is sampled by the environment's default randomiser, and the agent solves each episode from scratch. Terminated environments are auto-reset until N episodes have completed, and the returned dictionary contains `success_rate`, per-episode success flags, and the seeds used. This mode is appropriate when the benchmark itself defines the distribution of initial states, goals, and termination conditions, and it is the natural protocol for environments whose goals are generated procedurally at reset time or for robustness sweeps where FoV options and wrappers are applied directly to the live simulator.

Dataset-driven Evaluation. `world.evaluate(dataset=`$\mathcal{D}$`, episodes_idx=[`i_1`,...], start_steps=[`t_1`,...], goal_offset=`Δ`, eval_budget=`B`)` runs the dataset-conditioned protocol used in DINO-WM [22], LeWM [21], and PLDM [19]. Instead of sampling an arbitrary initial state and goal online, both are sampled from an existing trajectory: if a dataset trajectory is written as $\tau^i = (o_0^i, \dots, o_T^i)$, evaluation initialises the environment at o_t^i and uses $o_{t+\Delta}^i$ as the target observation. The offset Δ controls how far the goal lies from the start, and the dataset split controls the evaluation distribution. Because the target was observed later in the same trajectory, the goal is reachable by construction under the recorded behaviour, which removes a major ambiguity in goal-conditioned planning benchmarks: failure can be interpreted as a planning or modelling failure rather than as an impossible start–goal pair.

This setting is useful for comparing world models under controlled distribution shift. The same policy or planner can be evaluated on expert training trajectories, held-out expert trajectories, random-policy trajectories, or trajectories collected under FoV perturbations, while keeping the start–goal construction fixed. Experiments can vary reachability horizon and distribution shift independently: `goal_offset` changes the temporal distance to the goal, and the chosen dataset changes the visual, behavioural, or physical regime from which starts and goals are drawn.

Budget. Both protocols cap each episode at `eval_budget` environment steps. The cap keeps wall-clock cost comparable across methods that use different planners, control frequencies, or per-step computation, regardless of whether one method (e.g. MPPI with large `num_samples`) consumes more per-step compute than another.

Reported metrics. For each (method, environment, FoV-setting) we report success rate, time-to-goal among successful episodes, and wall-clock latency per planning step. The gap between dataset-driven and episodic success rate serves as a proxy for over-reliance on in-distribution behavioural coverage [58].

F Planning Solvers

In `swm`, solvers are responsible for computing the optimal action that the policy should execute. To do this, `swm` implements a suite of planning solvers that optimize action sequences by leveraging a world model to evaluate costs. Specifically, all solvers receive the current latent state $\boldsymbol{s}_0$ and use a world model $(\mathcal{E}_\theta, \mathcal{P}_\theta)$ to evaluate the cost of an action sequence $A = (\boldsymbol{a}_0, \dots, \boldsymbol{a}_{H-1})$ over a planning horizon H:

$$J_\theta(\boldsymbol{s}_0, A) = \sum_{t=0}^{H-1} c(\boldsymbol{s}_t, \boldsymbol{a}_t), \tag{2}$$

where $\boldsymbol{s}_{t+1} = \mathcal{P}_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)$, and $c(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is a task-specific goal-reaching cost. Each solver's objective is to return an action sequence $A^\star$ that minimizes J_θ:

$$A^\star = \arg\min_A J_\theta(\boldsymbol{s}_0, A). \tag{3}$$

The implemented solvers cover two broad families of approaches, sampling-based (zeroth-order) and gradient-based (first-order), and also span both continuous and discrete action regimes. Each solvers' algorithm is described below. See Table 3 for a complete comparison of each solver.

F.1 Sampling-Based Solvers

Sampling-based methods iteratively draw populations of candidate action sequences. The candidates are evaluated by the cost J_θ, and refined by slowly moving toward low-cost sequences. These methods are zeroth-order: they only require forward evaluations of the world model and do not assume that J_θ is differentiable. This makes them broadly applicable across learned models, handcrafted costs, and environments with discontinuous or non-smooth objectives.

The main trade-off is the curse of dimensionality. Since sampling-based methods do not use gradient information, they often require a sufficiently large population to explore the action space. In `swm`, candidate rollouts are executed in parallel, making these methods effective when batched on GPU.

Table 3: Selecting a solver. "Diff. model required" means the solver back-propagates through `get_cost`.

Solver	Family	Action space	Diff. model required	Constraints
`PredictiveSamplingSolver`	sampling	continuous	no	continuous box
`CEMSolver`	sampling	continuous	no	continuous box
`ICEMSolver`	sampling	continuous	no	continuous box
`MPPISolver`	sampling	continuous	no	continuous box
`CategoricalCEMSolver`	sampling	discrete	no	categorical simplex
`GradientSolver`	gradient	continuous	yes	continuous box
`PGDSolver`	gradient	discrete	yes	simplex projection
`GRASP`	gradient	continuous	yes	continuous box
`LagrangianSolver`	gradient	continuous + ineq.	yes	differentiable inequalities

Predictive sampling Predictive Sampling [28] is a single-shot sampling algorithm that works by repeatedly perturbing a nominal action sequence with Gaussian noise, storing the cost of the perturbed sequences, and returning the lowest-cost candidate.

Unlike CEM, iCEM, or MPPI, Predictive Sampling does not fit a new sampling distribution and does not perform iterative refinement inside a solver call. Its optimization behavior comes from random search around the nominal plan and from repeated MPC replanning over time.

Algorithm 9: Predictive Sampling Solver

Input: World model cost J_θ, current state s_0, horizon H, number of candidates N, noise scale σ, optional nominal sequence A^{nom}
Output: Action sequence $A^\star = (a_0^\star, \dots, a_{H-1}^\star)$
$\bar{A} \leftarrow A^{\text{nom}}$ if provided, otherwise $0_{H \times d_a}$
$\epsilon_1 \leftarrow 0$
for $i \leftarrow 2$ **to** N **do**
  $\epsilon_i \sim \mathcal{N}(0, \sigma^2 I_{H d_a})$
for $i \leftarrow 1$ **to** N **do**
  $A_i \leftarrow \bar{A} + \epsilon_i$
  $C_i \leftarrow J_\theta(s_0, A_i)$
$i^\star \leftarrow \arg\min_{i \in \{1,\dots,N\}} C_i$
return $A^\star \leftarrow A_{i^\star}$

Cross-entropy method The Cross-Entropy Method (CEM) [23] maintains a sampling distribution over action sequences. In `swm`, this distribution is a diagonal Gaussian with mean μ^ℓ and coordinate-wise standard deviation σ^ℓ over the full planning horizon. At each CEM iteration, the solver samples candidate action sequences, evaluates them with J_θ, selects the E lowest-cost candidates as elites, and refits the Gaussian parameters to those elites.

This iterative refitting concentrates probability mass around low-cost regions of the action sequences. The mean controls exploitation, while the standard deviation controls exploration and shrinks as the elites converge. After the final iteration, the solver returns the final mean sequence.

Algorithm 10: Cross-Entropy Method Solver

Input: World model cost J_θ, current state s_0, horizon H, number of candidates N, iterations L, elites E, initial scale σ_0, optional initial sequence A^{init}
Output: Action sequence $A^\star = (a_0^\star, \dots, a_{H-1}^\star)$
$\mu^0 \leftarrow A^{\text{init}}$ if provided, otherwise $0_{H \times d_a}$
$\sigma^0 \leftarrow \sigma_0 \mathbf{1}$
for $\ell \leftarrow 0$ **to** $L - 1$ **do**

$\epsilon_1^\ell \leftarrow 0$
for $i \leftarrow 2$ **to** N **do**
  $\epsilon_i^\ell \sim \mathcal{N}(0, I_{Hd_a})$
for $i \leftarrow 1$ **to** N **do**
  $A_i^\ell \leftarrow \mu^\ell + \sigma^\ell \odot \epsilon_i^\ell$
  $C_i^\ell \leftarrow J_\theta(s_0, A_i^\ell)$
$\mathcal{E}^\ell \leftarrow$ Select top E elites that minimize $\{C_i^\ell\}_{i=1}^N$
$\mu^{\ell+1} \leftarrow \frac{1}{E} \sum_{i \in \mathcal{E}^\ell} A_i^\ell$
$\sigma^{\ell+1} \leftarrow \mathrm{Std}_{i \in \mathcal{E}^\ell} \left(A_i^\ell\right)$
return $A^\star \leftarrow \mu^L$

Model Predictive Path Integral Model Predictive Path Integral (MPPI) control [30] is another sampling-based optimizer, but instead of selecting a hard elite set and refitting a Gaussian, it updates the nominal sequence using a soft cost-weighted average of sampled candidates. Each candidate receives a weight proportional to $\exp\left(-\frac{C_i - C_{\min}}{\lambda}\right)$, where C_i is the predicted cost and λ is a temperature parameter. Lower temperatures place most of the weight on the best candidates, while higher temperatures average over a broader set of samples.

In `swm`, MPPI samples candidates around a nominal mean sequence and updates only the mean, while the sampling scale remains fixed. It also supports applying the MPPI weighting to only the top-k candidates, which interpolates between MPPI-style soft weighting and more elite-focused behavior as in CEM. This makes MPPI a smooth alternative to CEM: rather than discarding non-elite candidates completely, it assigns them exponentially smaller influence based on their cost.

Algorithm 11: Model Predictive Path Integral Solver

Input: Cost J_θ, current state s_0, horizon H, number of candidates N, iterations L, top-k size E, temperature λ, sampling scale σ_0, optional initial sequence A^{init}
Output: Action sequence $A^\star = (a_0^\star, \ldots, a_{H-1}^\star)$
$\mu^0 \leftarrow A^{\text{init}}$ if provided, otherwise $0_{H \times d_a}$
$\sigma \leftarrow \sigma_0 \mathbf{1}$
for $\ell \leftarrow 0$ **to** $L - 1$ **do**
  $\epsilon_1^\ell \leftarrow 0$
  for $i \leftarrow 2$ **to** N **do**
    $\epsilon_i^\ell \sim \mathcal{N}(0, I_{Hd_a})$
  for $i \leftarrow 1$ **to** N **do**
    $A_i^\ell \leftarrow \mu^\ell + \sigma \odot \epsilon_i^\ell$
    $C_i^\ell \leftarrow J_\theta(s_0, A_i^\ell)$
  $\mathcal{E}^\ell \leftarrow$ Select top E that minimize $\{C_i^\ell\}_{i=1}^N$
  $C_{\min}^\ell \leftarrow \min_{i \in \mathcal{E}^\ell} C_i^\ell$
  for $i \in \mathcal{E}^\ell$ **do**
    $w_i^\ell \leftarrow \dfrac{\exp\left(-(C_i^\ell - C_{\min}^\ell)/\lambda\right)}{\sum_{j \in \mathcal{E}^\ell} \exp\left(-(C_j^\ell - C_{\min}^\ell)/\lambda\right)}$
  $\mu^{\ell+1} \leftarrow \sum_{i \in \mathcal{E}^\ell} w_i^\ell A_i^\ell$
return $A^\star \leftarrow \mu^L$

Improved Cross-Entropy Method Improved CEM (iCEM)[29] is based on CEM and adds colored-noise sampling, elite retention iterations, and momentum on distribution updates to give better sample efficiency, especially under small planning budgets.

Colored noise produces smoother action sequences by correlating perturbations across time, rather than sampling each action independently. This is useful for control tasks where good behavior often requires planning over several steps. Elite retention reuses a small number of strong candidates from

the previous CEM iteration, preventing useful samples from being discarded too quickly. Momentum smooths the updates to μ^ℓ and σ^ℓ, reducing instability when the elite set is small.

Algorithm 12: Improved Cross-Entropy Method Solver

Input: World model cost J_θ, current state s_0, horizon H, number of candidates N, iterations L, elites E, initial scale σ_0, colored-noise exponent β, momentum α, retained elites E_{keep}, action bounds $\mathcal{A}$, optional initial sequence A^{init}
Output: Action sequence $A^\star = (a_0^\star, \dots, a_{H-1}^\star)$
$\mu^0 \leftarrow A^{\text{init}}$ if provided, otherwise $\mathbf{0}_{H \times d_a}$
$\sigma^0 \leftarrow \sigma_0 \mathbf{1}$
$\mathcal{M}^0 \leftarrow \emptyset$
for $\ell \leftarrow 0$ **to** $L-1$ **do**
  $\xi_1^\ell \leftarrow 0$
  for $i \leftarrow 2$ **to** N **do**
    $\xi_i^\ell \sim \mathcal{C}_\beta(H, d_a)$
  for $i \leftarrow 1$ **to** N **do**
    $A_i^\ell \leftarrow \mu^\ell + \sigma^\ell \odot \xi_i^\ell$
  if $\ell > 0$ **then**
    $r \leftarrow \min(E_{\text{keep}}, |\mathcal{M}^\ell|)$
    Replace $A_2^\ell, \dots, A_{1+r}^\ell$ with the best r retained elites from $\mathcal{M}^\ell$
  for $i \leftarrow 1$ **to** N **do**
    $A_i^\ell \leftarrow \text{clip}_{\mathcal{A}}(A_i^\ell)$
    $C_i^\ell \leftarrow J_\theta(s_0, A_i^\ell)$
  $\mathcal{E}^\ell \leftarrow$ Select top E elites that minimize $\{C_i^\ell\}_{i=1}^N$
  $\mathcal{M}^{\ell+1} \leftarrow \{A_i^\ell : i \in \mathcal{E}^\ell\}$, ordered by increasing cost
  $\hat{\mu}^\ell \leftarrow \frac{1}{E} \sum_{i \in \mathcal{E}^\ell} A_i^\ell$
  $\hat{\sigma}^\ell \leftarrow \text{Std}_{i \in \mathcal{E}^\ell}\left(A_i^\ell\right)$
  $\mu^{\ell+1} \leftarrow \alpha \mu^\ell + (1-\alpha)\hat{\mu}^\ell$
  $\sigma^{\ell+1} \leftarrow \alpha \sigma^\ell + (1-\alpha)\hat{\sigma}^\ell$
return $A^\star \leftarrow \mu^L$

Categorical CEM. Categorical CEM is the discrete-action analog of CEM. Instead of maintaining a Gaussian distribution over continuous action sequences, it maintains an independent categorical distribution over the discrete action set at each planning step. Candidate sequences are sampled from these categorical distributions, converted to one-hot action representations, and evaluated with J_θ. The elite candidates are then used to update the categorical probabilities through empirical frequencies, optionally with smoothing and momentum. This solver is useful when actions are discrete and a continuous relaxation is not desired (in contrast to PGD).

Algorithm 13: Categorical Cross-Entropy Method Solver

Input: World model cost J_θ, current state s_0, discrete action set $\mathcal{A}$, horizon H, action blocks B_{act}, number of candidates N, iterations L, elites E, smoothing δ, momentum α
Output: Discrete action sequence $A^\star = (a_0^\star, \dots, a_{H-1}^\star)$
Initialize categorical probabilities $\pi_{t,b}^0(a) \leftarrow 1/|\mathcal{A}|$ for all $t = 0, \dots, H-1$, $b = 1, \dots, B_{\text{act}}$, and $a \in \mathcal{A}$
for $\ell \leftarrow 0$ **to** $L-1$ **do**

for $i \leftarrow 1$ **to** N **do**
  Sample $A_i^\ell = \{a_{i,t,b}^\ell\}_{t,b}$ from π^ℓ using Gumbel-max sampling
Set the first candidate to the current mode:
$a_{1,t,b}^\ell \leftarrow \arg\max_{a \in \mathcal{A}} \pi_{t,b}^\ell(a)$
for $i \leftarrow 1$ **to** N **do**
  $Y_i^\ell \leftarrow \mathrm{OneHot}(A_i^\ell)$
  $C_i^\ell \leftarrow J_\theta(s_0, Y_i^\ell)$
$\mathcal{E}^\ell \leftarrow$ indices of the E lowest costs in $\{C_i^\ell\}_{i=1}^N$
Refit categorical probabilities from elite frequencies:
$\hat{\pi}_{t,b}^\ell(a) \leftarrow \frac{1}{E} \sum_{i \in \mathcal{E}^\ell} \mathbf{1}[a_{i,t,b}^\ell = a]$
if $\delta > 0$ **then**
  $\hat{\pi}_{t,b}^\ell(a) \leftarrow \frac{\hat{\pi}_{t,b}^\ell(a) + \delta}{1 + |\mathcal{A}|\delta}$
Update probabilities with momentum: $\pi^{\ell+1} \leftarrow \alpha\pi^\ell + (1-\alpha)\hat{\pi}^\ell$
$a_{t,b}^\star \leftarrow \arg\max_{a \in \mathcal{A}} \pi_{t,b}^L(a)$
return $A^\star$

F.2 Gradient-Based Solvers

Gradient-based solvers work by directly optimizing the action sequence A with respect to the world model cost J_θ via first-order gradient information. Importantly, because they require J_θ to be differentiable, the world model predictor $\mathcal{P}_\theta$ must also be differentiable. Gradient-based solvers can be more sample-efficient than sampling-based methods because each update gives a directed improvement signal.

The main limitation of gradient-based solvers is that learned world models can have poorly conditioned gradients, especially over long horizons. Back-propagating through multiple predicted steps can produce unstable updates and trap the optimizer in local minima. `swm` includes simple baselines, such as Gradient Descent and PGD, as well as more structured methods like GRASP that modify the planning objective to improve optimization.

Gradient descent The gradient descent (GD) is a first-order algorithm that optimizes an action sequence by iteratively minimizing the differentiable world model cost J_θ. The gradient with respect to A is repeatedly applied to the action sequence. The solver initializes one or more candidate sequences and repeatedly updates each candidate in the direction of lower cost. After the final gradient step, it evaluates the optimized candidates and returns the lowest-cost sequence. The implementation can use standard optimizers such as Adam, optional gradient clipping, and multiple random initial candidates.

Algorithm 14: Gradient Descent Solver

Input: Differentiable world model cost J_θ, current state s_0, horizon H, number of candidates N, iterations K, step size η
Output: Action sequence $A^\star = (a_0^\star, \ldots, a_{H-1}^\star)$
Initialize candidates $\{A_i^0\}_{i=1}^N$, with $A_i^0 \in \mathbb{R}^{H \times d_a}$
for $k \leftarrow 0$ **to** $K-1$ **do**
  for $i \leftarrow 1$ **to** N **do**
    $A_i^{k+1} \leftarrow A_i^k - \eta \nabla_{A_i^k} J_\theta(s_0, A_i^k)$
$i^\star \leftarrow \arg\min_{i \in \{1,\ldots,N\}} J_\theta(s_0, A_i^K)$
return $A^\star \leftarrow A_{i^\star}^K$

Projected Gradient Descent Projected Gradient Descent (PGD) extends gradient-based planning to discrete action spaces by optimizing a continuous relaxation of the discrete actions. Instead of choosing a discrete action directly at each timestep, the solver optimizes a probability vector over the action set $\mathcal{A}$. Thus, the optimization variables lie in a product of simplexes, $P \in \Delta(\mathcal{A})^H$.

At each step, PGD takes a gradient step on the relaxed action probabilities and then projects each probability vector back onto the simplex using the projection algorithm of Duchi et al. [70]. This guarantees that every relaxed action remains a valid distribution over discrete actions. After optimization, the relaxed sequence is converted back to a discrete action sequence, either by taking the maximum probability action at each timestep or by sampling from the optimized distributions.

Algorithm 15: Projected Gradient Descent Solver

Input: Differentiable world model cost J_θ, current state s_0, discrete action set $\mathcal{A}$, horizon H, number of candidates N, iterations K, step size η, initial scale σ_0, optional action-noise scale σ_{act}, optional initial sequence A^{init}
Output: Discrete action sequence $A^\star = (a_0^\star, \dots, a_{H-1}^\star)$
$\bar{P} \leftarrow \text{OneHot}(A^{\text{init}})$ if provided, otherwise $\bar{P} \leftarrow \frac{1}{|\mathcal{A}|}\mathbf{1} \in \Delta(\mathcal{A})^H$
$P_1^0 \leftarrow \bar{P}$
for $i \leftarrow 2$ **to** N **do**
  $\epsilon_i \sim \mathcal{N}(0, \sigma_0^2 I_{H|\mathcal{A}|})$
  $P_i^0 \leftarrow \Pi_{\Delta(\mathcal{A})^H}\left(\bar{P} + \epsilon_i\right)$
for $k \leftarrow 0$ **to** $K-1$ **do**
  for $i \leftarrow 1$ **to** N **do**
    $\xi_i^k \leftarrow 0$
    if $\sigma_{\text{act}} > 0$ **then**
      $\xi_i^k \sim \mathcal{N}(0, \sigma_{\text{act}}^2 I_{H|\mathcal{A}|})$
    $P_i^{k+1} \leftarrow \Pi_{\Delta(\mathcal{A})^H}\left(P_i^k - \eta \nabla_P J_\theta(s_0, P_i^k) + \xi_i^k\right)$
$i^\star \leftarrow \arg\min_{i \in \{1,\dots,N\}} J_\theta(s_0, P_i^K)$
for $t \leftarrow 0$ **to** $H-1$ **do**
  $a_t^\star \leftarrow \arg\max_{a \in \mathcal{A}}\left(P_{i^\star,t,a}^K\right)$
return $A^\star \leftarrow (a_0^\star, \dots, a_{H-1}^\star)$

Lagrangian The Lagrangian solver extends gradient-based planning to inequality-constrained objectives. In addition to a differentiable cost J_θ, the world model may expose constraint functions $g_j(s_0, A) \le 0$. The solver then optimizes an augmented Lagrangian objective of the form

$$\mathcal{L}(A, \lambda, \rho) = J_\theta(s_0, A) + \sum_j \lambda_j g_j(s_0, A) + \rho \sum_j \max(0, g_j(s_0, A))^2.$$

The inner loop performs gradient-based optimization of the action sequence, while the outer loop updates the nonnegative multipliers λ_j by dual ascent and increases the penalty coefficient ρ. This solver is useful when a task includes explicit differentiable constraints, such as action-norm limits, safety margins, and other user-defined conditions.

Algorithm 16: Lagrangian Solver

Input: Differentiable cost J_θ, differentiable constraints g_θ, current state s_0, horizon H, number of candidates N, outer iterations L, inner gradient steps K, optimizer step size η, initial sampling scale σ_0, penalty parameters $\rho_0, \rho_{\max}, \rho_{\text{scale}}$, optional action-noise scale σ_{act}, optional initial sequence A^{init}
Output: Action sequence $A^\star = (a_0^\star, \dots, a_{H-1}^\star)$
Initialize nominal sequence $\bar{A} \leftarrow A^{\text{init}}$ if provided, otherwise $0_{H \times d_a}$
$A_1 \leftarrow \bar{A}$
for $i \leftarrow 2$ **to** N **do**
  $\epsilon_i \sim \mathcal{N}(0, \sigma_0^2 I_{Hd_a})$
  $A_i \leftarrow \bar{A} + \epsilon_i$
Initialize multipliers $\lambda^0 \leftarrow 0$ and penalty $\rho^0 \leftarrow \rho_0$
for $\ell \leftarrow 0$ **to** $L-1$ **do**

for $k \leftarrow 0$ **to** $K-1$ **do**
  for $i \leftarrow 1$ **to** N **do**
    $C_i \leftarrow J_\theta(s_0, A_i)$
    $G_i \leftarrow g_\theta(s_0, A_i)$
  Form the augmented Lagrangian objective:
  $\mathcal{L}_{\text{aug}} \leftarrow \sum_{i=1}^{N} \left[C_i + \lambda^\ell \cdot G_i + \rho^\ell \left\| [G_i]_+ \right\|_2^2 \right]$
  Update all candidate action sequences:
  $A_i \leftarrow A_i - \eta \nabla_{A_i} \mathcal{L}_{\text{aug}}, \quad i = 1, \ldots, N$
  if $\sigma_{\text{act}} > 0$ **then**
    for $i \leftarrow 1$ **to** N **do**
      $\xi_i \sim \mathcal{N}(0, \sigma_{\text{act}}^2 I_{Hd_a})$
      $A_i \leftarrow A_i + \xi_i$
  Re-evaluate constraint violations $G_i \leftarrow g_\theta(s_0, A_i)$ for $i = 1, \ldots, N$
  Update Lagrange multipliers by dual ascent:
  $\lambda^{\ell+1} \leftarrow \left[\lambda^\ell + \rho^\ell \frac{1}{N} \sum_{i=1}^{N} G_i \right]_+$
  Increase the quadratic penalty:
  $\rho^{\ell+1} \leftarrow \min\left(\rho_{\max}, \rho_{\text{scale}} \rho^\ell\right)$
for $i \leftarrow 1$ **to** N **do**
  $C_i \leftarrow J_\theta(s_0, A_i)$
$i^\star \leftarrow \arg\min_{i \in \{1,\ldots,N\}} C_i$
return $A^\star \leftarrow A_{i^\star}$

GRASP GRASP [32] jointly optimizes an action sequence and intermediate virtual states $z_1, \ldots, z_{H-1}$ as additional optimization variables. For each state z_{t+1}, it tries to minimize distance to the goal state, while not deviating from the state value predicted by the world model $\hat{z}_{t+1}^k \leftarrow \mathcal{P}_\theta\left(\text{sg}(z_t^k), a_t^k\right)$. Periodically, the implementation synchronizes the action sequence with the standard rollout cost J_θ, refining the current plan under the original MPC objective. Because the one-step dynamics terms can be evaluated in parallel across time, GRASP is especially useful for longer-horizon settings where ordinary rollout-based gradient descent becomes difficult to optimize.

Algorithm 17: GRASP Solver

Input: Encoder E_θ; predictor $\mathcal{P}_\theta$; World model cost J_θ; current observation o_0; goal observation o_g; horizon H; iterations K; action/state step sizes η_a, η_s; goal weights $\{\gamma_k\}_{k=0}^{K-1}$; state-noise scales $\{\sigma_k\}_{k=0}^{K-1}$; sync interval K_{sync}; sync operator Sync_{J_θ}; optional initial sequence A^{init}
Output: Action sequence $A^\star = (a_0^\star, \ldots, a_{H-1}^\star)$
$s_0 \leftarrow E_\theta(o_0), \quad s_g \leftarrow E_\theta(o_g)$
$A^0 \leftarrow A^{\text{init}}$ if provided, otherwise $0_{H \times d_a}$
for $t \leftarrow 1$ **to** $H-1$ **do**
  $z_t^0 \leftarrow \left(1 - \frac{t}{H}\right) s_0 + \frac{t}{H} s_g$
Fix $z_0^k \equiv s_0$ and $z_H^k \equiv s_g$
for $k \leftarrow 0$ **to** $K-1$ **do**
  Predict each transition in parallel:
  $\hat{z}_{t+1}^k \leftarrow \mathcal{P}_\theta\left(\text{sg}(z_t^k), a_t^k\right), \qquad t = 0, \ldots, H-1$
  $\mathcal{L}_k \leftarrow \sum_{t=0}^{H-1} \left(\left\| \hat{z}_{t+1}^k - z_{t+1}^k \right\|_2^2 + \gamma_k \left\| \hat{z}_{t+1}^k - s_g \right\|_2^2 \right)$
  $A^{k+1} \leftarrow A^k - \eta_a \nabla_{A^k} \mathcal{L}_k;$
  $z_{1:H-1}^{k+1} \leftarrow z_{1:H-1}^k - \eta_z \nabla_{z_{1:H-1}^k} \mathcal{L}_k;$
  Add stochastic state exploration:
  $\xi_t^k \sim \mathcal{N}(0, I)$
  $z_t^{k+1} \leftarrow z_t^{k+1} + \sigma_k \xi_t^k, \qquad t = 1, \ldots, H-1$
  if $K_{\text{sync}} > 0$ and $k > 0$ and $(k+1) \bmod K_{\text{sync}} = 0$ **then**
    $A^{k+1} \leftarrow \text{Sync}_{J_\theta}\left(A^{k+1}; s_0, s_g\right)$

return $A^\star \leftarrow A^K$

G Scripts & Command Line Interface

To further streamline the daily research workflow, `swm` provides a comprehensive suite of self-contained scripts alongside a dedicated CLI.

Scripts. Rather than requiring users to write repetitive boilerplate from scratch, the platform comes with ready-to-use, Hydra-configured scripts that cover the entire experimental pipeline [71]. Specifically, we provide scripts for efficient multimodal data collection across any supported environment, as well as scripts to train SAC and TD-MPC2 expert policies for generating high-quality demonstration data. In addition, `swm` includes standardised scripts to train and evaluate established baselines, ranging from imitation learning (GCBC) and offline RL (IQL, IVL) to modern world models (e.g. TD-MPC2, DINO-WM, PLDM, LeWM).

Command-Line Interface. The `swm` CLI lets researchers audit their workspace and data directly from the terminal, without writing any Python code. It exposes the following commands:

- `swm datasets` and `swm inspect`: report metadata on saved trajectories, including disk footprint, episode lengths, and the exact tensor shapes of stored columns.
- `swm envs` and `swm fovs`: list all available environments and display their customisable factors of variation.
- `swm checkpoints`: view and manage locally cached model weights.

H Extending `stable-worldmodel`

`swm` is meant as a community effort. As an open-source project, we strongly encourage contributions: new environments, new world models, or new planning solvers. We also plan to extend the benchmarking of included baselines and maintain a public leaderboard, so the state of the art in WM research is clearly tracked, and practitioners are guided in their choice of baselines.

New environment. Implement the standard `gymnasium.Env` interface and register the class under the `swm/` namespace. To unlock factor-of-variation studies, additionally expose a hierarchical `variation_space` built from `swm`'s extended spaces (`Box`, `Discrete`, `RGBBox`, `Dict`, etc.), each supporting an `init_value` and an optional `constrain_fn` for rejection sampling. The `info` dictionary returned by `reset` and `step` should carry any extra signals the user wishes to log or evaluate against; standard Gymnasium quantities (observation, action, termination, reward, frames) are recorded automatically. Once registered, the environment is immediately usable through `World`, the `swm` CLI, and every built-in evaluation routine.

New planning solver. A solver is any object exposing `configure(action_space, n_envs, config)` and `solve(info_dict, init_action)`, the latter returning a dict containing the optimized action sequence. New solvers are added by dropping a single file under `stable_worldmodel/solver/`.

New baseline. New baselines should ship as self-contained code, together with a training entry point under `scripts/train/`, mirroring the existing baselines (e.g. `pldm.py`, `gcivl.py`). At evaluation time, the baseline plugs into MPC through the `Costable` protocol: a single method `get_cost(info_dict, action_candidates) -> Tensor` of shape `(B, S)`.

I Experimental setup

This section details the specific configurations used in the experiments of Sec. 4. We follow the dataset-driven evaluation protocol described in App. E (`evaluate_from_dataset`), which specifies how start observations and goal observations are constructed from existing trajectories; here we focus only on the concrete settings.

Tasks and dataset. All experiments are conducted on the Push-T benchmark using the expert dataset released with DINO-WM [22]. Following the protocol of LeWM [21], for each evaluation start and goal observations are drawn from the same trajectory with a fixed temporal offset of $\Delta = 25$ environment steps. Distribution-shift sweeps additionally vary the source dataset (expert train, expert validation, random-policy, random-policy with FoV) while keeping Δ fixed.

Evaluation budget and metrics. Each episode is capped at $B = 50$ environment steps budget and we report success rate as defined in DINO WM [22]. In-distribution comparisons (Tab. 1) use 50 evaluation trajectories; the prediction-vs-success analyses (Fig. 4, 10) use 256 trajectories per regime for stable distributional estimates.

Planning. Unless otherwise noted, the planner is CEM (App. F, Alg. 10) with $L = 30$ iterations, $N = 300$ candidates, $E = 30$ elites, and initial sampling scale $\sigma_0 = 1$. World models are trained with a frameskip of 5; planning therefore uses a horizon of $H = 5$ model steps (25 environment steps) and is executed in open-loop mode, i.e. the full optimized action sequence is rolled out before replanning. Each planning experiment runs in a few minutes on a single L40S GPU thanks to `swm`'s efficient implementation.

Baselines. The robustness analyses (Tab. 5b, Fig. 11, Fig. 5a) primarily compare PLDM [19] and LeWM [21], which combine strong in-distribution planning performance with low per-step planning overhead, making systematic FoV sweeps tractable. DINO-WM [22] is additionally reported in Tab. 5b. Training hyperparameters for each baseline follow the original publications; the exact configs are released alongside the code

J Additional Results

This section presents supplementary plots from the experiments in Section 4. All results follow the standardized evaluation protocol of `stable-worldmodel` on the Push-T benchmark unless otherwise noted.

Table 4 provides a highly granular breakdown of the zero-shot generalization experiments by isolating individual factors of variation. Rather than reporting a single aggregate metric for "out-of-distribution" performance, this disentangled view reveals exactly which physical or visual properties break the learned representations. For instance, we observe that altering the target anchor's position or the agent's size drastically reduces planning success across all baselines. The models exhibit varying sensitivities: PLDM is remarkably robust to agent shape variations (52.0% success rate) compared to the others, but fails heavily when the agent color is perturbed.

The data loading benchmarks for the Two-Room environment are presented in Figure 8, confirming that our I/O optimizations generalize across different task structures and observation shapes. The results show that the Lance format maintains superior throughput, processing over 5.0k samples per second locally and 3.4k samples per second when streaming from a remote S3 bucket. While the MP4 format achieves the lowest disk storage footprint (220 MB), it severely degrades random access since retrieving a later frame often requires decoding all preceding frames in the clip. Lance thus provides the optimal trade-off between disk compression and high-speed random access.

Figures 9 and 10 expand on the prediction-vs-control analysis. Figure 9 shows that the per-trajectory MSE distributions for both PLDM and LeWM flatten and shift toward higher errors as the evaluation regime drifts from training, moving from expert train to expert validation, then to random policy, and finally to random policy with full variations. Figure 10 provides the corresponding success/failure breakdown for PLDM: even under strong distribution shift, the MSE densities of successful and failed rollouts overlap heavily, so a low prediction error does not guarantee planning success.

Figure 11 provides a continuous view of this brittleness by reporting LeWM's planning success rate across the chromatic wheel. Performance stays high near the center (close to the white background) and along the green axis, which matches the color of the target anchor in the unperturbed Push-T environment. As the background shifts toward red, blue, or purple at higher intensities, the success rate collapses, suggesting that the model latches onto specific background-foreground color contrasts rather than the underlying task geometry.

Table 4: Planning success rates (%) under individual factors of variation on Push-T evaluated on random-policy OOD trajectories.

FoV	Entity	SR % (↑)		
		LeWM	PLDM	DINO-WM
None		**50.8**	**50.8**	**20.0**
Color	Anchor	14.0	10.0	20.0
	Agent	12.0	8.0	18.0
	Block	22.0	18.0	18.0
	Canvas	6.0	6.0	10.0
Size	Anchor	26.0	18.0	14.0
	Agent	22.0	18.0	4.0
	Block	20.0	18.0	16.0
Angle	Anchor	20.0	24.0	12.0
	Agent	14.0	22.0	12.0
Position	Anchor	32.0	18.0	4.0
Shape	Agent	26.0	52.0	18.0
	Block	12.0	14.0	8.0
Velocity	Agent	16.0	16.0	14.0

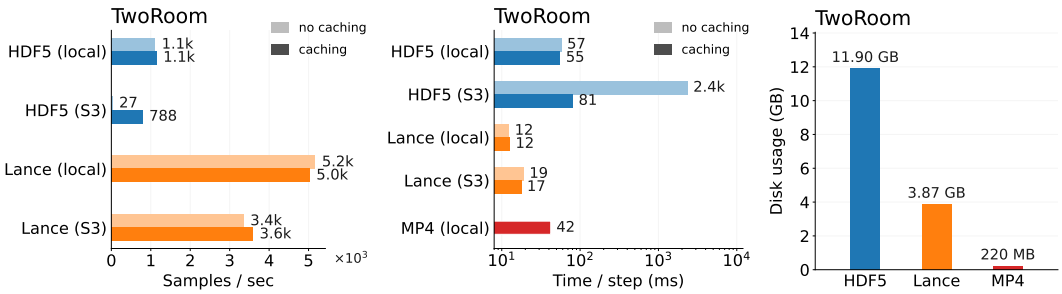
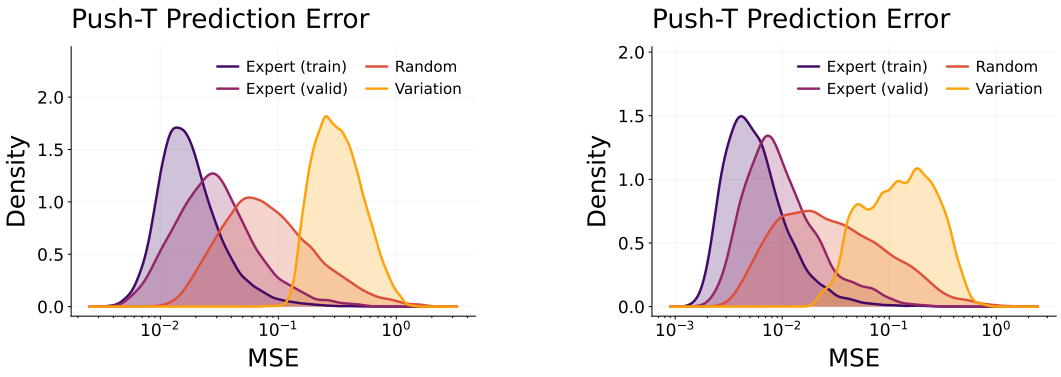


Figure 8: Performance comparison of different data formats for a dataset from the Two-Room environment. **(Left)** Data loading throughput (samples/sec) with and without caching, for both local storage and remote (S3) streaming. **(Center)** Per-step loading latency (ms). **(Right)** Disk storage usage. Results demonstrate that Lance is the most efficient format in terms of throughput and fetch latency while remaining a good trade-off in compression for storage.

Finally, Tab. 5 and Fig. 12 validate the continuous control solvers in the online regime. While Sec. 4.1 reports that TD-MPC2 struggles to plan effectively offline on Push-T due to out-of-distribution action sampling, these results show strong online performance: across several DeepMind Control Suite tasks, TD-MPC2 reaches competitive asymptotic rewards and matches the sample efficiency of a Soft Actor-Critic (SAC) baseline. This confirms that the algorithm is correctly implemented within `swm` and that its offline limitations are algorithmic rather than an implementation artifact. Fig. 13 illustrates this offline failure mode directly: a PCA projection of the latent state space shows that, unlike expert rollouts which stay within the training distribution, trajectories produced by TD-MPC2's actor quickly drift away from the data manifold, fooling the predictor and leading to poor planning performance.

K Research Opportunities

`stable-worldmodel` is intended as an open platform for world-model research and evaluation. In the remainder of this section, we outline several promising research directions that `swm` makes immediately tractable for the community.



(a) PLDM (b) LeWM

Figure 9: Push-T prediction error under increasing distribution shift for **(a)** PLDM and **(b)** LeWM . Density of per-trajectory MSE across four regimes of growing OOD severity: expert train, expert validation, random-policy, and random-policy with all `swm` factors of variation jointly perturbed. Both models exhibit an error increase as the evaluation set shifts from training.

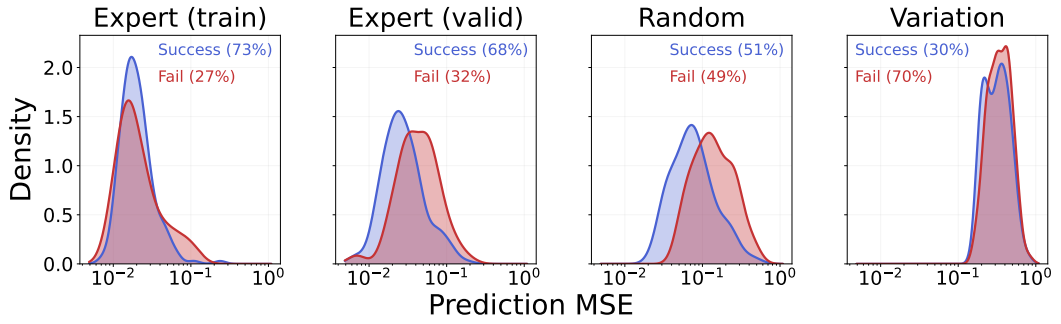


Figure 10: PLDM counterpart of Fig. 4. Distribution of trajectory-level prediction MSE for successful (blue) and failed (red) plans on Push-T, across four regimes of increasing distribution shift (expert train, expert validation, random-policy, and random-policy with all `swm` factors of variation). Evaluated on 256 trajectories per regime. As with LeWM, the success and failure distributions overlap heavily even under a strong shift, confirming that prediction error is a poor predictor of planning success.

Achieving performant zero-shot world models. Out-of-distribution generalization and zero-shot evaluation are central motivations behind `swm`, and the platform directly exposes the tools needed to study them: factors of variation, environment wrappers, and a unified baseline suite. As shown in Sec. 4, current approaches remain brittle even under simple visual perturbations such as color shifts of the scene. Closing this gap on visual and physical robustness is required for safe real-world deployment of world models, yet it will demand substantial further progress. We hope `swm` lowers the barrier for the community to pursue it.

Unlocking long-horizon planning. Reliable long-horizon planning remains a major limitation of current world models: as we have shown in Fig. 5, prediction errors compound over rollouts, and most published evaluations rely on short horizons sampled from in-distribution trajectories. `swm` ships with a broad set of planning solvers and reference baselines, together with goal sampling at controllable temporal offsets, making it straightforward to stress-test models well beyond the short, expert-policy regimes used today. We hope this would motivate research on horizon scaling, hierarchical, and multi-step planning.

Scaling World Models. Scaling laws have driven much of the recent progress in language modeling [72]. Whether analogous regularities and their associated emergent behaviours hold for world models is an open and exciting question. `swm` is designed with this regime in mind: its high-throughput Lance-based data layer alleviates the I/O bottlenecks that typically starve accelerators during multi-

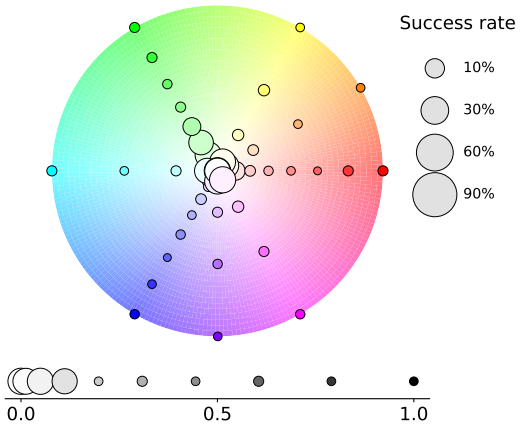
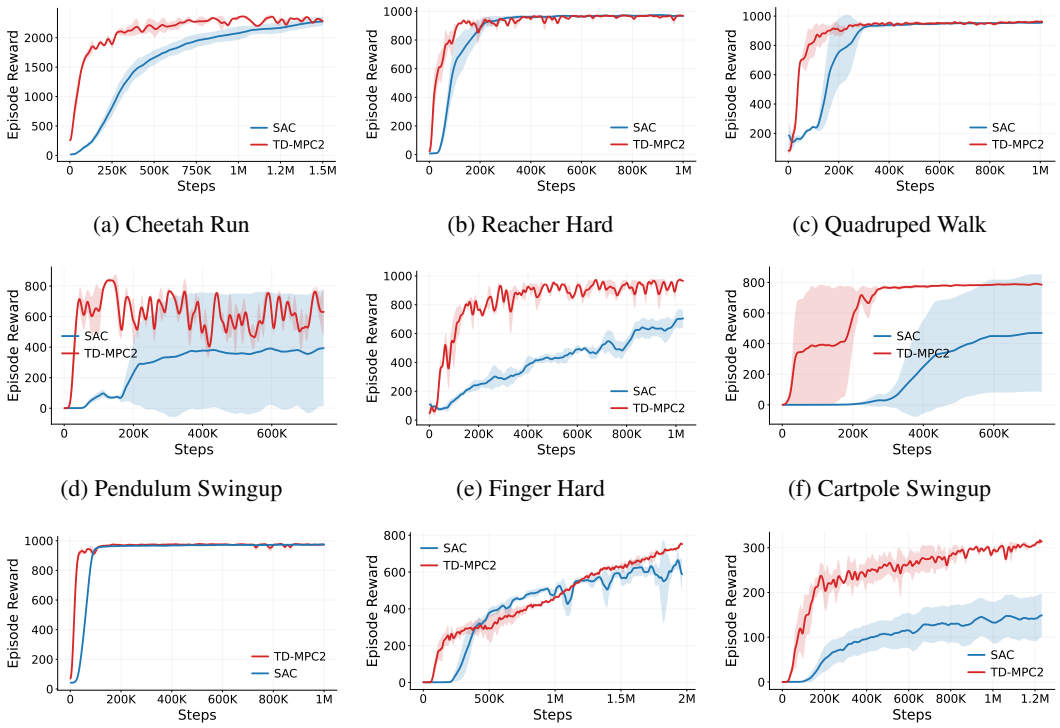


Figure 11: Planning success rate of LeWM on Push-T as a function of background color intensity along the chromatic wheel. Performance remains high near the original white background and is notably more robust to green backgrounds, likely due to the green visual anchor present in the default environment. Success degrades sharply for most other colors.

Table 5: Online training rewards on DeepMind Control Suite tasks. Mean and standard deviation computed over 25 evaluation episodes.

Environment	**TD-MPC2**		**SAC**	
	Mean ± Std	**Min / Max**	**Mean ± Std**	**Min / Max**
cartpole_swingup	$\mathbf{767 \pm 1}$	766 / 768	764 ± 4	760 / 766
cheetah_run	$\mathbf{2397 \pm 9}$	2356 / 2405	2369 ± 20	2308 / 2387
finger_turn_hard	$\mathbf{897 \pm 262}$	0 / 1000	893 ± 264	0 / 1000
hopper_hop	$\mathbf{355 \pm 8}$	340 / 370	265 ± 16	222 / 284
humanoid_walk	$\mathbf{749 \pm 30}$	685 / 800	543 ± 283	2 / 898
pendulum_swingup	$\mathbf{870 \pm 86}$	686 / 1000	839 ± 91	698 / 988
quadruped_walk	956 ± 24	876 / 990	$\mathbf{959 \pm 22}$	893 / 992
reacher_hard	969 ± 12	943 / 997	$\mathbf{976 \pm 13}$	939 / 994
walker_walk	$\mathbf{977 \pm 9}$	960 / 990	975 ± 19	893 / 994

modal training, as shown in Fig. 3, while its standardized collection and evaluation pipelines make it straightforward to compare models across data, parameter, and compute scales under identical protocols. We hope this turns scaling studies of world models into a concrete research agenda.



(a) Cheetah Run (b) Reacher Hard (c) Quadruped Walk

(d) Pendulum Swingup (e) Finger Hard (f) Cartpole Swingup

(g) Walker Walk (h) Humanoid Walk (i) Hopper Hop

Figure 12: Benchmarking `stable-worldmodel` against standard baselines across diverse environments. Solid lines depict the mean episode reward over 5 random seeds, while shaded areas denote the standard deviation. TD-MPC2 consistently achieves faster convergence in continuous control tasks.

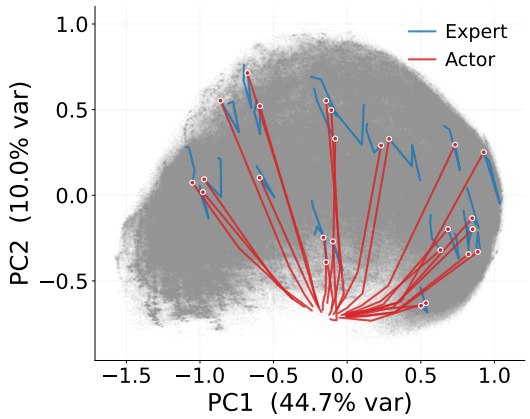


Figure 13: PCA projection of TD-MPC2's latent state space on Push-T. Gray points show the distribution of expert states from the offline dataset. Blue trajectories are rolled out from the expert policy and remain within the data manifold, while red trajectories are produced by TD-MPC2's actor and quickly drift outside the support of the training distribution. This visualizes the out-of-distribution drift that drives the planner's poor offline performance reported in Sec. 4.1.