stable-worldmodel-V1: REPRODUCIBLE WORLD MODELING RESEARCH AND EVALUATION [TINY PAPER]

Anonymous authors
Paper under double-blind review

ABSTRACT

World Models have emerged as a powerful paradigm for learning compact, predictive representations of environment dynamics, enabling agents to reason, plan, and generalize beyond direct experience. Despite recent interest in World Models, most available implementations remain publication-specific, severely limiting their reusability, increasing the risk of bugs, and reducing evaluation standardization. To mitigate these issues, we introduce `stable-worldmodel` (SWM), a modular, tested, and documented world-model research ecosystem that provides efficient data-collection tools, standardized environments, planning algorithms, and baseline implementations. In addition, each environment in SWM enables controllable factors of variation, including visual and physical properties, to support robustness and continual learning research. Finally, we demonstrate the utility of SWM by using it to study zero-shot robustness in DINO-WM.

– World Model Research Made Simple.

1 INTRODUCTION

A promising paradigm toward building capable and general-purpose embodied agents involves learning dynamics models of the world, commonly referred to as World Models (WM, Ha & Schmidhuber (2018)).

Despite rapid progress and growing community interest, research on WMs remains fragmented and lacks shared benchmarks comparable to those in vision (Russakovsky et al., 2015; Lin et al., 2014), reinforcement learning (Bellemare et al., 2013; Brockman et al., 2016; Tassa et al., 2018), or language modeling (Wang et al., 2024; Phan et al., 2025). This diversity of paradigms, design choices, and environments complicates meaningful comparison between methods. Systematic re-implementation of utilities further exacerbates this issue: for example, two recent works, PLDM (Sobal et al., 2025) and DINO-WM(Zhou et al., 2025), re-implement the same Two-Room environment with substantial divergence (81 deletions, 86 additions, and 18 updates), underscoring the lack of shared infrastructure.

Moreover, beyond comparing performance across disparate environments, controlled variations within a single environment are essential to isolate key factors, probe generalization, and better understand the inductive biases and failure modes of WMs.

In this work, we introduce `stable-worldmodel`, a new research ecosystem designed to facilitate streamlined and reproducible experimentation and benchmarking WMs. We design a simple, easy-to-use API that allows custom dataset collection, training, and evaluation, as well as integration of novel algorithms and environments to support future growth and development. A comparison with other recent latent world model codebases is provided in Table 1.

2 STABLE WORLD MODEL ECOSYSTEM: AN OVERVIEW

Stable World Model (SWM) goal is to support researchers by reducing the idea-to-experiment time gap. We build the library around the philosophy that people already have their codebase or tool for training their model. Therefore, our library should focus on providing support for their training with a ready-to-use environment and utilities for data collection or model evaluation. In the rest of this

Table 1: **Latent World-Model codebases comparison.** (PR = Pull Request, LoC = Lines of Code) Collected statistics demonstrate the lack of a reliable, open-source, and unified codebase to perform world model research. We address this issue with our proposed library SWM.

	SWM (ours)	**PLDM**	**DINO-WM**
Backend	PyTorch	PyTorch	PyTorch
Documentation	✓	✗	✗
# Baselines	4	1	1
# Environments	16	2	4
# FoV (per env)	6-17	0	0
Type Checking	✓	✓	✗
Test Coverage	73%	0%	0%
Last Commit	<1 week	>3 months	>10 months
PRs (6 mo.)	99	1	0
# LoC	3562	6796	4349

section, we provide an overview of the user API and the different components of the library. A full overview of a typical world model pipeline with SWM is provided in Listing 3.

2.1 THE WORLD INTERFACE: STREAMLINED WM RESEARCH

```python
import stable_worldmodel as swm

world = swm.World('swm/PushT-v1', num_envs=8)
world.set_policy(YourExpertPolicy())

world.reset() # initialize the world
world.step()  # update the world state with policy
world.infos   # current world state (dict)
```

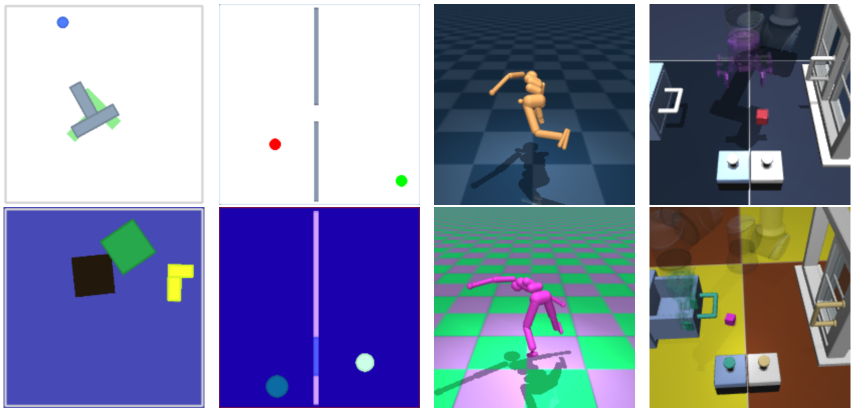
Listing 1: **World Interface Logic.** After specifying the environment ID (e.g., `swm/PushT-v1`) and the number of simulations, a policy can be attached to enable online interaction with the environment. At any time, all simulation-related information can be accessed via the `infos` dictionary.

The core abstraction in SWM is the `World`. A `World` wraps one or more Gymnasium **?** environments and provides a unified interface for simulation, data collection, debugging, and evaluation. Internally, it leverages Gymnasium's synchronous environment API to manage and step multiple environments within a single object.

Unlike the widely used Gymnasium (Towers et al., 2025) interface, a `World` does not return observations, rewards, or termination flags from `reset` or `step`. Instead, all data produced by the environments is stored in a single internal dictionary, `world.infos`, which is updated in place at every reset or step. Both methods operate synchronously over all environments, making the complete simulation state accessible at any time via `world.infos`.

Action selection in SWM is handled by a policy object attached to the `World`. The `step` method does not take actions as input; instead, at each step, the world queries its policy to obtain actions for all environments. A policy is a lightweight Python object implementing a `get_action` method, which takes the current `world.infos` as input and returns one action per environment. This design cleanly decouples control logic from environment execution, allowing policies to be swapped without modifying the world interface.

Once a policy is attached to a `World`, it can be used to record datasets or perform evaluation. Dataset recording executes the policy over episodes and logs all information contained in `world.infos`, while evaluation runs the same execution loop without data persistence. In both cases, the behavior and properties of the resulting trajectories are entirely determined by the chosen policy and world configuration. An illustrative example of dataset recording is provided in Listing 2. Additional details about the dataset and evaluation are reported in Appendix B.



(a) PushT (b) TwoRoom (c) DMC – Humanoid (d) OGBench – Scene

Figure 1: SWM **Environment Suite.** We support (and extend) a diverse set of established environments, including 2D/3D settings with tasks in manipulation, navigation, and classic control. (a) Push-T (Chi et al., 2025). A manipulation task where a blue agent needs to push a T-shaped block to match the green anchor. (b) Two-Room (Sobal et al., 2025). A 2d navigation task where a red agent needs to navigate through a door to reach a green goal in the room. (c) DeepMind Control Suite (Tassa et al., 2018), a collection of 3d control tasks in MuJoCo. (d) OGBench (Park et al., 2025), a 3D robotic manipulation task collection in MuJoCo. (Top) Default settings. (Bottom) All factors of variations changing visual, geometric, and physical properties. All supported environments and their associated FoV can be found in Figure 2 and Table 3.

2.2 ENVIRONMENTS AND FACTOR OF VARIATIONS

SWM is designed as a collection of diverse environments that span a wide range of design choices, including continuous and discrete state/action spaces, different action modalities, and varied agent embodiments. These environments differ not only in their task structure but also in their underlying dynamics or observation spaces, as illustrated in Figure 1. Such diversity allows evaluation across qualitatively distinct settings and supports broad comparisons of learning algorithms. However, evaluating generalization solely across different environments can obscure more fine-grained sources of variation that commonly arise within a single task or domain.

A key feature of SWM is the notion of *factors of variation (FoV)*. Each environment in the library exposes a set of optional controllable properties that enable systematic customization of the environment configuration. These factors of variation span multiple aspects, including visual attributes (e.g., color, shape, textures, lighting), geometric properties (e.g., size, orientation, position), and physical parameters (e.g., friction, damping, mass, gravity). By explicitly exposing these controls, SWM enables fine-grained studies of robustness, generalization, domain shift, and continual learning within a single, unified environment. We provide a toy example in Listing 2. More details about FoV can be found in Appendix B

```
import stable_worldmodel as swm

world = swm.World('swm/PushT-v1', num_envs=2)
world.set_policy(YourExpertPolicy())

print(world.single_variation_space.names()) # available FoV

# dataset with changing all agent FoV, and T color.
world.record_dataset(
dataset_name='pusht_demo',episodes=4, seed=0,
options={"variation": ["agent", "block.color"]},
)
```

Listing 2: SWM **Factor of Variation Logic.** During data collection or world reset, factors of variation (FoV) can optionally be specified via the `options` argument. In this illustrative Push-T example, all agent-related FoVs (e.g., color and size) are sampled, along with the color of the T-shaped object.

Internally, FoVs are implemented as a new type of Gymnasium dictionary Space (in addition to the standard action and observation space), which stores an internal value that can be initialized, sampled with or without constraint.

2.3 SWM EVALUATION SUITE: TASKS, PLANNING ALGORITHMS, AND BASELINES

Evaluating world models is inherently challenging, as existing works rely on diverse evaluation settings. SWM provides built-in support for *goal-conditioned evaluation*, where the agent is tasked to reach a specified goal representation, such as a target state, image, or reward condition. Performance is measured in terms of success rate, defined as the percentage of evaluation episodes that end satisfying the goal condition.

In SWM, evaluation can be conducted through the `World` interface and applied to the currently attached policy. These methods are `evaluate` and `evaluate_from_dataset`. SWM is agnostic to the choice of policy. Yet, we provided some utilities to facilitate planning with Model Predictive Control (MPC) (Richalet et al., 1978) or Feed-Forward action prediction. We provided further details on the specifics of each evaluation method and different MPC solvers in Appendix B.

3 EXPERIMENTS: DINO-WM ZERO-SHOT ROBUSTNESS

We now demonstrate how SWM can be used as a research tool to analyze model robustness. Specifically, we leverage SWM to evaluate the robustness of our reproduction of DINO-WM (Zhou et al., 2025) under both in-distribution and out-of-distribution evaluation settings, as well as its zero-shot generalization to environmental variations (e.g., agent color and background) in the Push-T environment. First, we observe that although DINO-WM performs well when evaluated on expert demonstrations, achieving a success rate of 94.0%, its performance deteriorates sharply under distribution shift. When evaluated on reaching states drawn from trajectories collected by a random policy, the success rate drops to 12.0%, revealing a strong dependence on the provenance of evaluation data. Next, using SWM as a controlled evaluation framework, we probe DINO-WM's zero-shot robustness to a range of factors of variation, as summarized in Table 2. Across all tested perturbations, the model exhibits consistently low scores, indicating limited robustness to unseen environmental variations despite the task structure remaining unchanged.

Table 2: **DINO-WM robustness on Push-T.** Zero-shot success rates (SR) across factors of variation (FoV) not seen during training, revealing pronounced sensitivity to shifts in environmental properties.

FoV	**Property**	**SR % (↑)**
Color	Anchor	20.0
	Agent	18.0
	Block	18.0
	Background	10.0
Size	Anchor	14.0
	Agent	4.0
	Block	16.0
Angle	Anchor	12.0
	Agent	12.0
Position	Anchor	4.0
Shape	Agent	18.0
	Block	8.0
Velocity	Agent	14.0

4 CONCLUSION AND FUTURE DIRECTIONS

With a streamlined API SWM promotes standardized evaluation, which we hope will accelerate progress in world-model research. We plan some future updates focusing on tools for improving debugging and interpretation of world models. Moreover, we will work on adding new environment support to the library with a focus on physical simulation or real-world tasks. Finally, our long-term vision aims to provide a standardized benchmark to keep track of the state-of-the-art in controllable world models, e.g., via a Hugging Face Benchmark.

REFERENCES

Randall Balestriero, Hugues Van Assel, Sami BuGhanem, and Lucas Maes. stable-pretraining-v1: Foundation model research made simple, 2025. URL https://arxiv.org/abs/2511.

19484.

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.

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

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

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

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Seohong Park, Kevin Frans, Benjamin Eysenbach, and Sergey Levine. OGBench: Benchmarking offline goal-conditioned RL. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=M992mjgKzI.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Jacques Richalet, André Rault, JL Testud, and J Papon. Model predictive heuristic control. *Automatica (journal of IFAC)*, 14(5):413–428, 1978.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Vlad Sobal, Wancong Zhang, Kyunghyun Cho, Randall Balestriero, Tim G. J. Rudner, and Yann LeCun. Stress-testing offline reward-free reinforcement learning: A case for planning with latent dynamics models. In *7th Robot Learning Workshop: Towards Robots with Human-Level Abilities*, 2025. URL https://openreview.net/forum?id=jON7H6A9UU.

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

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2025. URL https://arxiv.org/abs/2407.17032.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Gaoyue Zhou, Hengkai Pan, Yann LeCun, and Lerrel Pinto. Dino-wm: World models on pre-trained visual features enable zero-shot planning. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025)*, 2025.

A CODE EXAMPLE

A.1 END-TO-END PIPELINE.

```python
import stable_worldmodel as swm
from stable_worldmodel.data import HDF5Dataset
from stable_worldmodel.policy import WorldModelPolicy, PlanConfig
from stable_worldmodel.solver import CEMSolver

world = swm.World('swm/PushT-v1', num_envs=8)
world.set_policy(your_expert_policy)

#=== Record Dataset ===
world.record_dataset(
    dataset_name='pusht_demo',
    episodes=100,
    seed=0,
    options={"variation": ["all"]},
)

# ... train your world model with pusht_demo ...
world_model = ...  # your world-model implementing get_cost

#=== Evaluate World Model ===
dataset = HDF5Dataset(
    name='pusht_demo',
    frameskip=1,
    num_steps=16,
    keys_to_load=['pixels', 'action', 'state']
)

# model predictive control
solver = CEMSolver(model=world_model, num_samples=300, device='cuda')
policy = WorldModelPolicy(
    solver=solver,
    config=PlanConfig(horizon=10, receding_horizon=5)
)

world.set_policy(policy)
results = world.evaluate(episodes=50, seed=0)

print(f"Success Rate: {results['success_rate']:.1f}%")
```

Listing 3: stable-worldmodel pipeline example

A.2 POLICY

```python
import stable_worldmodel as swm
class MyPolicy:
    def get_action(self, info: dict) -> np.ndarray:
        """
        Args:
            info: dict with all information collected from the
               environments
        Returns:
            actions: Array of shape (num_envs, action_dim)
        """
        return actions

# Use policy
world = swm.World('swm/PushT-v1', num_envs=8)
world.set_policy(MyPolicy())
```

Listing 4: Policy definition and usage.

A.3 DATASET RECORDING

```
import stable_worldmodel as swm

world = swm.World('swm/PushT-v1', num_envs=8)
world.set_policy(YourExpertPolicy())

world.record_dataset(
dataset_name='pusht_demo',
episodes=100,
seed=0,
options={"variation": ["all"]},
)
```

Listing 5: SWM Data collection.

B SWM DETAILS

B.1 POLICY

Policy. Unlike Gymnasium, the `step` function does not take actions as an argument. Instead, actions are determined by a policy object associated with the world. At each call of the `step` method, the world queries the policy to obtain the actions for all environments. A policy is a simple Python object implementing a `get_action` method. This method receives the current world `infos` and returns an action for each environment. Decoupling action selection from the `step` call makes it easy to swap policies within a single script without modifying the world interface. We provide a boilerplate example for policy implementation and usage in Listing 4.

Model Predictive Control. SWM supports planning-based control by enabling world models to infer policies through the solution of a finite-horizon planning problem, i.e., optimizing the optimal sequence of actions reaching the goal. To this end, we provide a dedicated `MPCPolicy`. This policy is parameterized by a `PlanConfig`, which defines the Model Predictive Control (MPC) setup (e.g., planning horizon and receding horizon, warm start), and a `Solver` object responsible for optimizing the action sequence.

We re-implement several widely used planning solvers, including the Cross-Entropy Method (CEM), Model Predictive Path Integral (MPPI), and gradient-based optimizers (e.g., SGD, Adam). All solvers are implemented with efficiency and numerical stability in mind and are extensively tested to ensure reliability.

B.2 DATASET RECORDING

Once a world is created and a policy is attached, datasets can be collected using the `record_dataset` method. This API runs episodes by executing the policy associated with the world and records the resulting interactions and all information contained in the internal state of the world. As a result, the quality and characteristics of the collected data are entirely determined by the chosen policy and world configuration. By default, we save all datasets in the HDF5 format. Yet, we support other formats like image folders or mp4 videos for specific usage. An illustrative example of dataset recording is provided in Listing 2.

B.3 FACTOR OF VARIATIONS

FoVs are configured through an optional dictionary passed via the `options` argument at reset, dataset recording, or evaluation time. To enable variation, the `variations` key specifies a list of FoV names to be modified. We adopt a common hierarchical naming convention of the form `key_1.key_2` to reference FoVs within an environment. For example, `agent` applies variations to all agent-related properties, whereas `agent.color` restricts variation to the agent's color only. All FoV can be changed simultaneously by setting `variations` to `all` as illustrated in listing 5. By default, specified FoVs are resampled at each `reset`; however, fixed values can be enforced by providing explicit assignments through the `variation_values` key in `options`.

B.4 EVALUATIONS

In SWM, evaluation can be conducted under two complementary protocols, both accessible directly through the `World` interface and applied to the currently attached policy.

First, an *online* evaluation protocol samples (or allows the user to specify) both the initial state and the goal at the beginning of each episode, following prior work such as PLDM. This setting evaluates the policy through direct environment interaction and can be invoked using the world `evaluate` method.

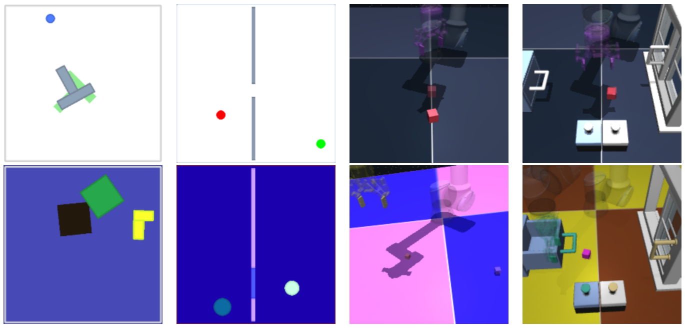
Alternatively, SWM supports an *offline* evaluation protocol. In this setting, a complete trajectory is first sampled from a specified dataset, typically collected using an expert policy. The initial state and goal are then selected from this trajectory subject to a constraint on the maximum number of steps separating them. This protocol guarantees that the task is feasible within a given step budget, enabling controlled and reliable evaluation of planning and model accuracy without additional environment interaction. This setting, similar to DINO-WM, can be invoked using the world `evaluate_from_dataset` method.

C EXPERIMENT DETAILS

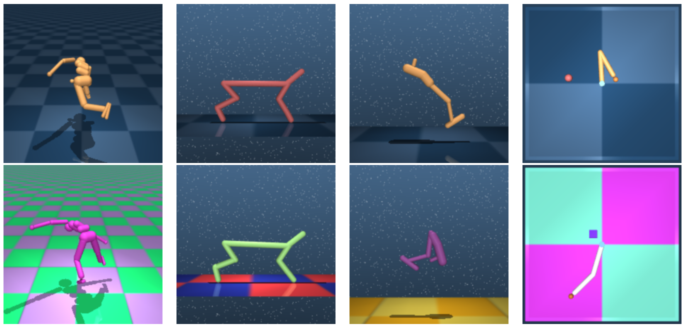
Training Details. Our re-implementation of DINO-WM has been implemented in PyTorch (Paszke et al., 2019) and trained with stable-pretraining (Balestriero et al., 2025). We train for 20 epochs with the same hyperparameters as those prescribed in the original publication.

Evaluation Details. We use the Cross Entropy Method (CEM) solver with the same set of parameters as the original DINO-WM publication. However, unlike the original work, which had an infinite planning budget, we fixed the steps budget to 50, which corresponds to 2x the minimum number of steps required to succeed (25).

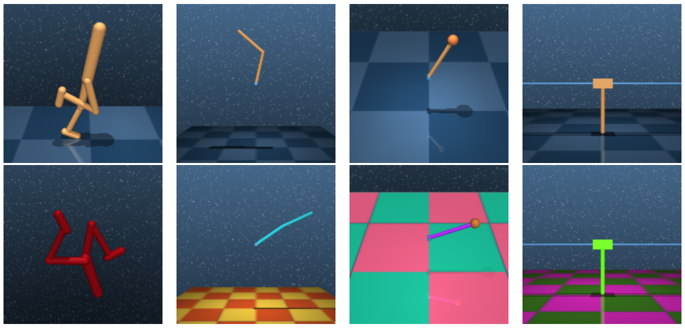
D SWM ENVIRONMENTS



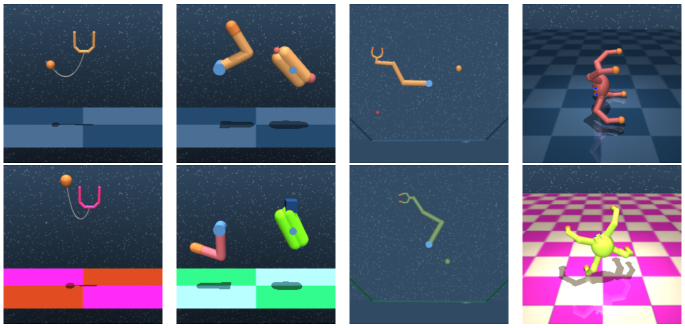
(a) PushT (b) TwoRoom (c) OGBench – Cube (d) OGBench – Scene



(e) Humanoid (f) Cheetah (g) Hopper (h) Reacher



(i) Walker (j) Acrobot (k) Pendulum (l) Cartpole



(m) Ball-in-Cup (n) Finger (o) Manipulator (p) Quadruped

Figure 2: **Visualization of** SWM **Environments suite.**

Environment ID	**# FoV**	**Available Variation(s)**
swm/PushT-v1	16	agent.angle, agent.color, agent.scale, agent.shape, agent.start_position, agent.velocity, background.color, block.angle, block.color, block.scale, block.shape, block.start_position, goal.angle, goal.color, goal.position, goal.scale
swm/TwoRoom-v1	17	agent.color, agent.max_energy, agent.position, agent.radius, agent.speed, background.color, door.color, door.number, door.position, door.size, goal.color, goal.position, goal.radius, wall.axis, wall.border_color, wall.color, wall.thickness
swm/OGBCube-v0	11	agent.color, agent.ee_start_position, camera.angle_delta, cube.color, cube.goal_position, cube.goal_yaw, cube.size, cube.start_position, cube.start_yaw, floor.color, light.intensity
swm/OGBScene-v0	12	agent.color, agent.ee_start_position, camera.angle_delta, cube.color, cube.goal_position, cube.goal_yaw, cube.size, cube.start_position, cube.start_yaw, floor.color, light.intensity, lock_color
swm/HumanoidDMControl-v0	7	agent.color, agent.left_knee_locked, agent.right_lower_arm_density, agent.torso_density, floor.color, floor.friction, light.intensity
swm/CheetahDMControl-v0	7	agent.back_foot_density, agent.back_foot_locked, agent.color, agent.torso_density, floor.color, floor.friction, light.intensity
swm/HopperDMControl-v0	7	agent.color, agent.foot_density, agent.foot_locked, agent.torso_density, floor.color, floor.friction, light.intensity
swm/ReacherDMControl-v0	8	agent.arm_density, agent.color, agent.finger_density, agent.finger_locked, floor.color, light.intensity, target.color, target.shape
swm/WalkerDMControl-v0	8	agent.color, agent.left_foot_density, agent.right_knee_locked, agent.torso_density, floor.color, floor.friction, floor.rotation_y, light.intensity
swm/AcrobotDMControl-v0	8	agent.color, agent.lower_arm_density, agent.upper_arm_density, agent.upper_arm_locked, floor.color, light.intensity, target.color, target.shape
swm/PendulumDMControl-v0	6	agent.color, agent.mass_density, agent.mass_shape, agent.pole_density, floor.color, light.intensity
swm/CartpoleDMControl-v0	6	agent.cart_mass, agent.cart_shape, agent.color, agent.pole_density, floor.color, light.intensity
swm/BallInCupDMControl-v0	9	agent.color, agent.density, ball.color, ball.density, ball.size, floor.color, light.intensity, target.color, target.shape
swm/FingerDMControl-v0	10	agent.color, agent.fingertip_density, agent.proximal_density, floor.color, light.intensity, spinner.color, spinner.density, spinner.friction, target.color, target.shape
swm/ManipulatorDMControl-v0	8	agent.color, agent.hand_density, agent.upper_arm_density, agent.upper_arm_length, floor.color, light.intensity, target.color, target.shape
swm/QuadrupedDMControl-v0	7	agent.color, agent.foot_back_left_density, agent.knee_back_left_locked, agent.torso_density, floor.color, floor.friction, light.intensity

Table 3: Summary of SWM environments, and controllable factor of variations.