Part One Report

Diffusion Policy Policy Optimization

by

Elton Chun-Chai Li

Submitted in partial fulfilment of the requirements for

Machine Learning Center of Excellence Summer Associate

in the

2025-2026

Date of submission: 08 December 2024

Table of Contents

1 Literature Review

1.1 Reinforcement Learning

Reinforcement learning (RL), particularly Deep Reinforcement Learning (DRL), has evolved to handle increasingly complex problems with high-dimensional input spaces. Within the model-free paradigm, three main approaches have emerged: value-based, policy-based, and actor-critic methods. Value-based methods, exemplified by DQN and its variants, focus on learning a value function that estimates expected future rewards. Policy-based methods, such as REINFORCE and Proximal Policy Optimization (PPO), directly optimize the policy to maximize expected rewards. Actor-critic methods, including Deep Deterministic Policy Gradient (DDPG) and Soft Actor-Critic (SAC), combine both approaches by using an actor to select actions and a critic to evaluate them.

Recently, there has been a growing trend toward incorporating diffusion models into reinforcement learning frameworks. This integration addresses several fundamental challenges in traditional RL approaches, particularly in terms of training stability and the ability to represent complex behavior patterns. The success of diffusion models in other domains of machine learning, combined with their potential to enhance RL capabilities, has sparked significant interest in this direction, particularly for applications requiring sophisticated behavior generation and robust performance.

1.2 Diffusion for Reinforcement Learning

Diffusion models have emerged as a powerful class of generative models, operating through a gradual denoising process. These models first add Gaussian noise to data in multiple steps until it becomes pure noise, then learn to reverse this process to generate new data. Two primary frameworks have evolved: the Denoising Diffusion Probabilistic Model (DDPM) with discrete steps, and Score-Based Generative Models extending to continuous-time diffusion processes.

The integration of diffusion models with RL has demonstrated success in three main applications:

1. As planners: Generating complete trajectories including states, actions, and rewards
2. As policies: Offering greater expressiveness than conventional Gaussian policies
3. As data synthesizers: Addressing data scarcity in RL applications

In robotics applications, diffusion-based policies have shown particular promise in modeling complex and multi-modal trajectory distributions, typically trained from human demonstrations. To overcome the limitations of demonstration data, researchers have developed various improvements, including:

- Guided diffusion denoising processes using reward signals
- Goal conditioning approaches
- Q-learning and weighted regression techniques in both offline and online settings

The advantages of diffusion models in RL include their ability to represent complex, multi-modal distributions, making them effective for sophisticated behavior patterns. However, these benefits come with computational costs, particularly in the sampling process and training resources. Despite these challenges, the integration of diffusion models with RL represents a promising direction, particularly in robotics applications where complex behavior modeling is crucial.

1.3 Fine-tuning Diffusion-Based Policies

Pre-trained diffusion policies, while powerful in their ability to model complex behaviors, face several critical limitations that stem primarily from their reliance on demonstration data. These policies are fundamentally constrained by the quality, quantity, and coverage of their training demonstrations. Expert demonstrations, while valuable, may be suboptimal or fail to encompass the full range of scenarios a policy might encounter in practice. This limitation becomes particularly apparent when policies face situations that deviate from their training distribution, leading to poor generalization and potentially catastrophic failures in real-world applications.

Fine-tuning emerges as a crucial solution by enabling policies to learn from direct interaction with their environment. This process allows policies to transcend the constraints of their initial training data, discovering more efficient strategies and adapting to new scenarios through reinforcement learning. By combining the structured knowledge captured in pre-training with the ability to learn from experience, fine-tuning creates policies that are both more robust and more capable of optimal performance. The benefits are particularly evident in practical applications, especially robotics, where policies can adapt to specific deployment conditions and bridge the gap between simulation and reality.

DPPO (Diffusion Policy Policy Optimization) represents a significant advancement in addressing the limitations of fine-tuning diffusion-based policies. At its core, DPPO introduces a novel two-layer "Diffusion Policy MDP" that embeds the denoising process MDP within the environmental MDP of control tasks. This framework proves particularly effective in handling challenges that previous methods struggled with, especially in scenarios with sparse rewards—a common characteristic in robotic manipulation tasks where success signals only appear upon completing specific objectives.

DPPO addresses several critical technical challenges that plagued earlier approaches. It tackles the inefficiency of updating the entire denoising chain by introducing two effective solutions: focusing updates on only the final few steps of the denoising process, or utilizing the Denoising Diffusion Implicit Model (DDIM) sampling technique during fine-tuning. These innovations substantially reduce the required steps without compromising performance. Additionally, DPPO introduces a novel approach to exploration by leveraging the inherent properties of the pre-trained diffusion model, ensuring exploration remains relevant to the task while allowing for the discovery of improved policies.

Empirical evidence strongly supports DPPO's effectiveness in practical applications. The method consistently outperforms other diffusion-based RL algorithms in benchmark tests, achieving higher success rates and faster convergence, particularly in challenging tasks with long horizons and sparse rewards. Moreover, DPPO demonstrates superior performance compared to conventional policy parameterizations like Gaussian and GMM, even when these are fine-tuned using PPO. Perhaps most impressively, DPPO has shown remarkable success in bridging the sim-to-real gap, facilitating the transfer of policies trained in simulation to real-world robotic systems. This capability has been demonstrated in complex tasks such as furniture assembly, where DPPO achieves high success rates on physical hardware without requiring real-data fine-tuning—a crucial advantage for practical robotics applications.

1.4 Summary

The evolution from traditional reinforcement learning to diffusion-based approaches represents a significant advancement in handling complex robotic tasks. Diffusion models have proven their value by offering better representation of complicated behavior patterns, particularly in scenarios requiring precise control and adaptation. The introduction of DPPO addresses a critical gap in this field by enabling effective fine-tuning of pre-trained diffusion policies. This advancement is particularly significant as it combines the benefits of demonstration learning with the adaptability of reinforcement learning. While diffusion models do require more computational resources compared to traditional approaches, their ability to handle complex, multi-modal action distributions and successfully transfer from simulation to real-world applications makes them particularly promising for practical robotics applications. The success of DPPO in tasks like furniture assembly demonstrates that diffusion-based policies can effectively bridge the gap between theoretical capabilities and real-world implementation, opening new possibilities for robust and adaptable robotic control systems.

2 Proposed Methods and Choices

DPPO represents a significant advancement in fine-tuning diffusion-based policies, and its replication holds substantial value for real-world applications. For this replication effort, TensorFlow will be the primary framework, motivated by both the task requirements and its robust ecosystem for production deployment. The choice of TensorFlow is particularly relevant for real-world deployment scenarios where production-ready implementations are essential.

The original work provides a comprehensive PyTorch implementation that serves as an excellent foundation for our replication efforts. This codebase includes well-documented installation procedures, execution guidelines, and extensive configuration options for different experimental settings. Furthermore, it offers a complete pipeline covering pre-training, fine-tuning, and evaluation phases, along with pre-trained checkpoints that will be invaluable for verification purposes. These resources significantly streamline the replication process and provide crucial reference points for validation.

Our implementation strategy will maintain the modular and configurable nature of the original codebase while transitioning to TensorFlow. Each critical module will be reimplemented with a "_tf" suffix, preserving the original architecture's structure and configuration system. This approach ensures that the flexibility and extensibility of the original implementation are maintained while leveraging TensorFlow's capabilities. The implementation will prioritize core DPPO algorithm components, including the denoising process, MDP structure, and policy gradient implementation, followed by the training and evaluation loops.

To ensure the reliability and correctness of our implementation, a comprehensive testing framework using pytest will be developed. This framework will enable direct comparisons with the PyTorch results, facilitate gradient checking, and allow for thorough numerical validation. The specific details of this testing strategy and validation approach will be further elaborated in Section 4, where we will discuss the verification process in detail.

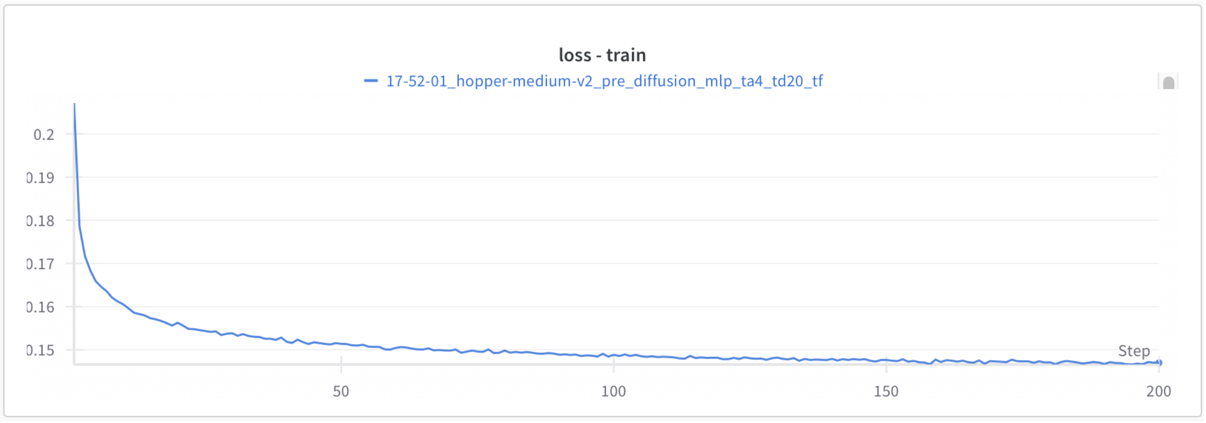
3 Results and Analysis

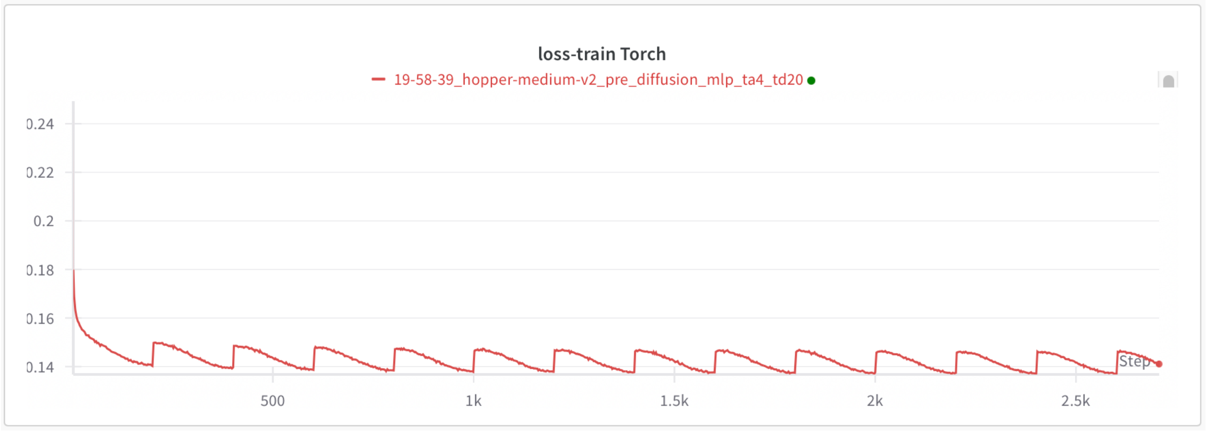
3.1 Replication Results with TensorFlow(TF)

The current implementation successfully replicates the basic functionality of DPPO in TensorFlow. Our implementation supports two key components: pre-training with diffusion MLP and fine-tuning experiments with DPPO on gym environments. While the core functionality is operational, additional features and environment support are planned for future implementation.

Pre-training Results

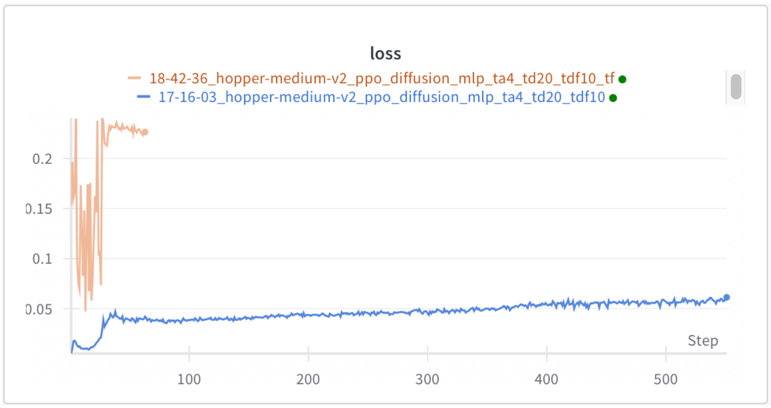
The pre-training experiments were conducted on the hopper-medium-v2 dataset, following the configuration parameters provided in the original demo. Due to computational resource constraints, the training was limited to 200 epochs rather than the full training regime specified in the paper. The loss curves between our TensorFlow implementation and the original PyTorch version show similar patterns during these initial epochs, suggesting correct basic functionality. Further extended training runs are planned to verify long-term convergence behavior.

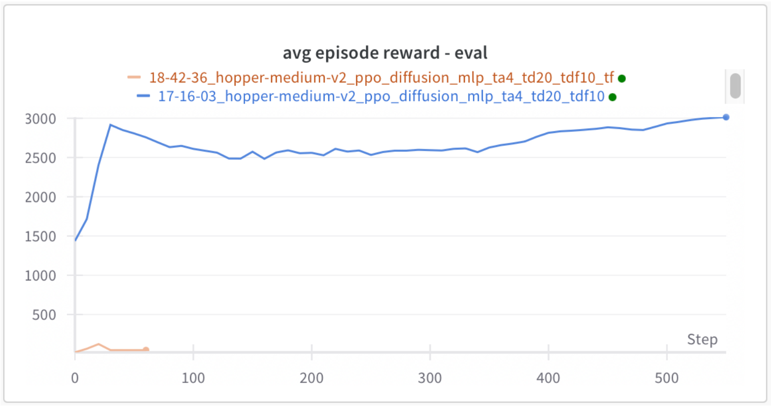


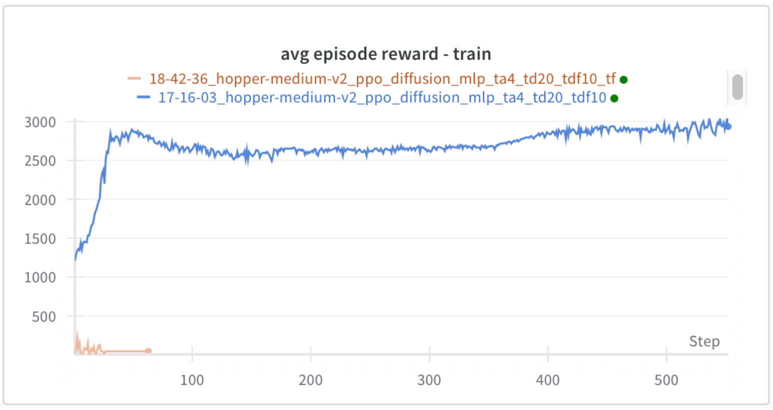


Fine-tuning

Fine-tuning experiments were performed using the model pre-trained for 200 epochs. However, significant discrepancies were observed between our TensorFlow implementation and the original PyTorch version. Key metrics including loss values and average episode rewards show substantial differences, indicating potential implementation issues. These differences suggest the presence of underlying bugs in our TensorFlow implementation that require attention. We have identified several areas for investigation: loss computation, policy gradient implementation details, reward scaling and normalization, and training loop. Further testing and debugging efforts will focus on identifying and resolving these discrepancies to achieve closer alignment with the original implementation's performance.





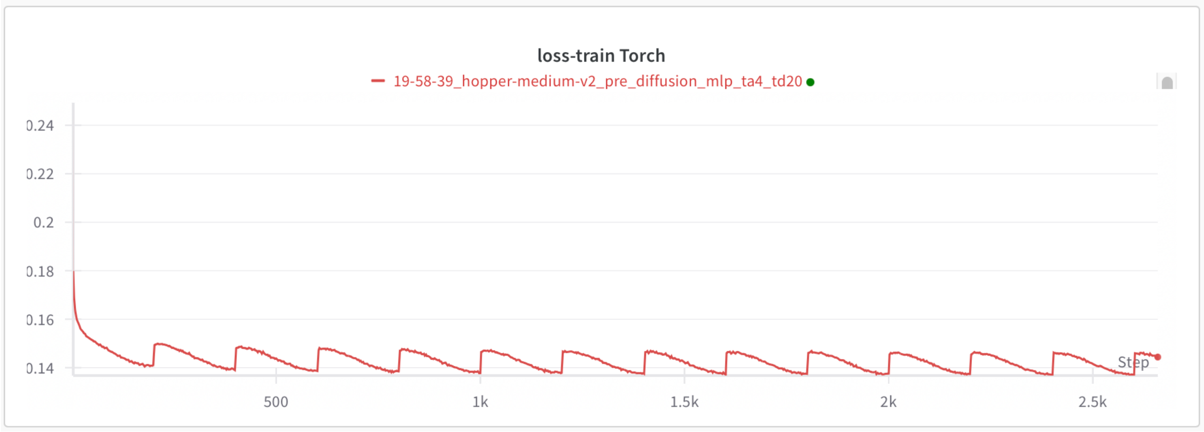


3.2 Unclear Components and Missing Details

During our implementation process, several components of the original work require further investigation and clarification. The optimizer implementation details and the cosine scheduler mechanism are two key areas that need additional study. These components play crucial roles in the training process, yet their exact implementations and parameter choices are not fully detailed in the original paper or codebase. Further investigation into these aspects will be conducted to ensure accurate replication of the original work's performance characteristics.

3.3 Implementation Error and Discrepancies

During our pre-training process, we encountered an interesting observation when running the original implementation's demo script. Using the provided PyTorch code on the hopper-medium-v2 dataset, we observed an unexpected zigzag pattern in the loss curve, which appears to differ from the smooth convergence behavior. This discrepancy in the original implementation's behavior warrants further investigation, as it might indicate either undocumented implementation details or specific running conditions that affect the training stability.



Due to computational resource constraints, our TensorFlow implementation's experiments are currently limited to 200 training epochs, making it challenging to fully validate the long-term convergence behavior. A comprehensive validation plan has been developed to investigate both implementations further. This will involve extended runs of both the original PyTorch implementation and our TensorFlow version, with particular attention to understanding the source of the zigzag pattern observed in the demo script's output. We plan to test various environmental conditions to determine whether this behavior is inherent to the implementation or due to specific running conditions. These findings will be crucial for ensuring the accuracy of our TensorFlow replication and understanding any fundamental behavioral characteristics of the DPPO algorithm not fully addressed in the original paper.

3.4 Critical Review

Comprehensive evaluation of the DPPO paper were done based on its technical contributions, empirical validation, and practical significance. The paper demonstrates several compelling strengths that make it a valuable contribution to the field. The extensive empirical validation across multiple environments (OpenAI Gym, Robomimic, and Furniture-Bench) provides strong evidence for the algorithm's effectiveness. Particularly noteworthy is DPPO's superior performance in challenging sparse-reward scenarios and complex manipulation tasks, achieving over 90% success rate in the Transport task from Robomimic. The successful sim-to-real transfer with an 80% success rate in real-world robot manipulation tasks is especially impressive, as it addresses a critical challenge in robotics applications. The paper's analysis is thorough and well-structured, offering clear insights into why DPPO works through its demonstration of structured exploration near the pre-training data manifold and the benefits of multi-step denoising process.

However, there are several aspects where the paper could be enhanced. The implementation details of certain components, such as the optimizer configuration and cosine scheduler, could be more thoroughly documented to facilitate reproduction. While the paper demonstrates DPPO's effectiveness, more detailed ablation studies examining the impact of different architectural choices and hyperparameters would strengthen the understanding of the algorithm's key components. Additionally, the paper would benefit from a more detailed discussion of the computational resources required for training and deployment, which is crucial for practical applications.

Based on our analysis, we would recommend accepting this paper. The strong empirical results, particularly in real-world robotics applications, represent a significant advancement in the field. The comprehensive evaluation across multiple environments and the successful sim-to-real transfer demonstrate both theoretical soundness and practical utility. The paper's approach to combining diffusion models with PPO is novel and well-executed, offering a promising direction for future research in policy learning. While there are areas for improvement in documentation and analysis, these do not diminish the paper's substantial contributions to the field of reinforcement learning and robotics. The success in challenging, sparse-reward environments and real-world robotics applications makes this work particularly valuable to the research community and practitioners in the field.

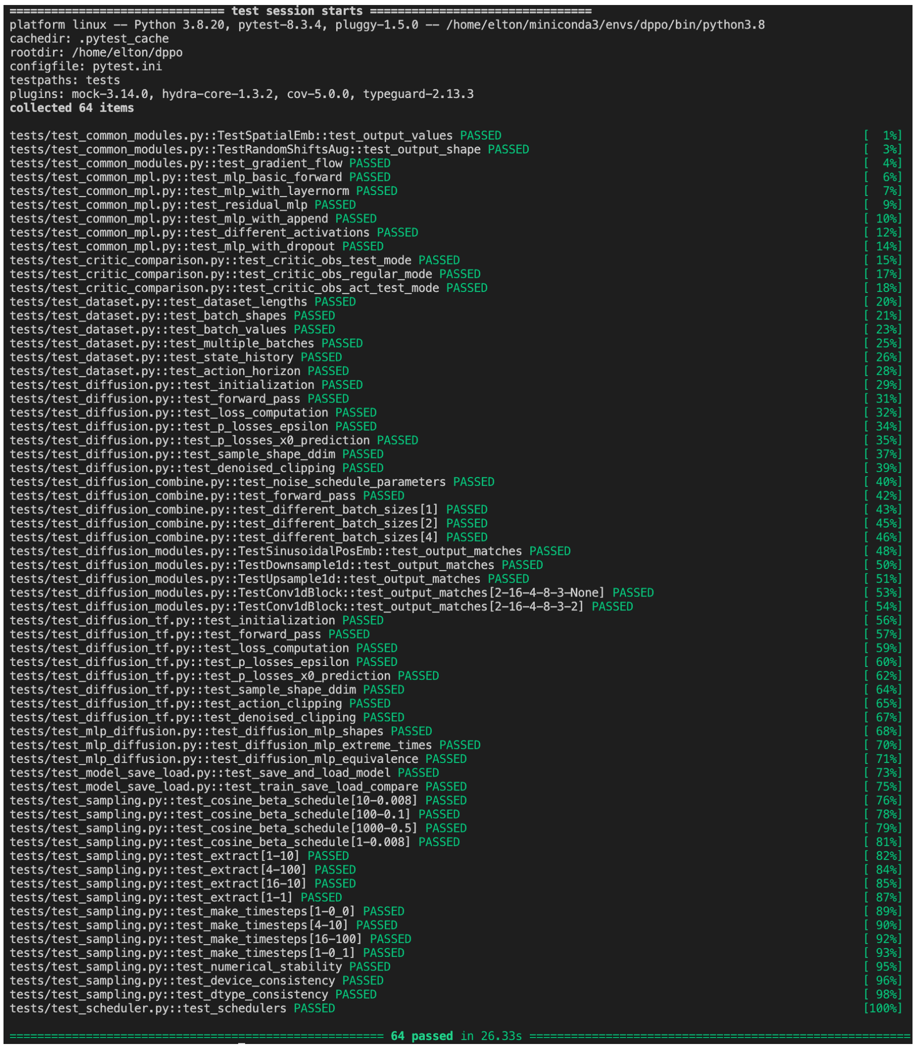
4 Testing Plan

To ensure system reliability and accurate replication of the DPPO algorithm, we have implemented a multi-layered testing strategy encompassing unit tests, regression tests, and integration tests. Our testing framework is designed to verify both individual components and their interactions within the larger system. Our unit tests focus on verifying the correctness of fundamental components, including the sampler, cosine scheduler, policy network, loss functions, and various utility functions. These tests ensure that each component operates correctly in isolation, with particular attention to numerical accuracy and computational efficiency.

Integration testing forms a crucial part of our validation strategy, examining the interaction between different system components. These tests verify the correct assembly of the complete DPPO architecture, the functionality of the full training pipeline (including data loading, forward passes, and optimization steps), proper checkpoint management, and memory usage patterns during extended training sessions. To ensure faithful replication, we conduct comparative testing between our TensorFlow implementation and the original PyTorch version, including direct comparison of model outputs given identical inputs, verification of gradient computation and backpropagation patterns, and comparison of training metrics and model behavior.

Several additional testing components are planned for future implementation to enhance the robustness and reliability of our system. These include comprehensive tests for the fine-tuning pipeline, with particular focus on policy optimization verification, reward calculation accuracy, and environment interaction testing.

Currently, while our test cases comprehensively cover the pre-training pipeline for gym environments, future work will expand this coverage to include the complete fine-tuning pipeline and additional environmental interactions.



```
=============================== test session starts ================================
platform linux -- Python 3.8.20, pytest-8.3.4, pluggy-1.5.0 -- /home/elton/miniconda3/envs/dppo/bin/python3.8
cachedir: .pytest_cache
rootdir: /home/elton/dppo
configfile: pytest.ini
testpaths: tests
plugins: mock-3.14.0, hydra-core-1.3.2, cov-5.0.0, typeguard-2.13.3
collected 64 items

tests/test_common_modules.py::TestSpatialEmb::test_output_values PASSED                             [  1%]
tests/test_common_modules.py::TestRandomShiftsAug::test_output_shape PASSED                         [  3%]
tests/test_common_modules.py::test_gradient_flow PASSED                                             [  4%]
tests/test_common_mpl.py::test_mlp_basic_forward PASSED                                             [  6%]
tests/test_common_mpl.py::test_mlp_with_layernorm PASSED                                            [  7%]
tests/test_common_mpl.py::test_residual_mlp PASSED                                                  [  9%]
tests/test_common_mpl.py::test_mlp_with_append PASSED                                               [ 10%]
tests/test_common_mpl.py::test_different_activations PASSED                                         [ 12%]
tests/test_common_mpl.py::test_mlp_with_dropout PASSED                                              [ 14%]
tests/test_critic_comparison.py::test_critic_obs_test_mode PASSED                                   [ 15%]
tests/test_critic_comparison.py::test_critic_obs_regular_mode PASSED                                [ 17%]
tests/test_critic_comparison.py::test_critic_obs_act_test_mode PASSED                               [ 18%]
tests/test_dataset.py::test_dataset_lengths PASSED                                                  [ 20%]
tests/test_dataset.py::test_batch_shapes PASSED                                                     [ 21%]
tests/test_dataset.py::test_batch_values PASSED                                                     [ 23%]
tests/test_dataset.py::test_multiple_batches PASSED                                                 [ 25%]
tests/test_dataset.py::test_state_history PASSED                                                    [ 26%]
tests/test_dataset.py::test_action_horizon PASSED                                                   [ 28%]
tests/test_diffusion.py::test_initialization PASSED                                                 [ 29%]
tests/test_diffusion.py::test_forward_pass PASSED                                                   [ 31%]
tests/test_diffusion.py::test_loss_computation PASSED                                               [ 32%]
tests/test_diffusion.py::test_p_losses_epsilon PASSED                                               [ 34%]
tests/test_diffusion.py::test_p_losses_x0_prediction PASSED                                         [ 35%]
tests/test_diffusion.py::test_sample_shape_ddim PASSED                                              [ 37%]
tests/test_diffusion.py::test_denoised_clipping PASSED                                              [ 39%]
tests/test_diffusion_combine.py::test_noise_schedule_parameters PASSED                              [ 40%]
tests/test_diffusion_combine.py::test_forward_pass PASSED                                           [ 42%]
tests/test_diffusion_combine.py::test_different_batch_sizes[1] PASSED                               [ 43%]
tests/test_diffusion_combine.py::test_different_batch_sizes[2] PASSED                               [ 45%]
tests/test_diffusion_combine.py::test_different_batch_sizes[4] PASSED                               [ 46%]
tests/test_diffusion_modules.py::TestSinusoidalPosEmb::test_output_matches PASSED                   [ 48%]
tests/test_diffusion_modules.py::TestDownsample1d::test_output_matches PASSED                       [ 50%]
tests/test_diffusion_modules.py::TestUpsample1d::test_output_matches PASSED                         [ 51%]
tests/test_diffusion_modules.py::TestConv1dBlock::test_output_matches[2-16-4-8-3-None] PASSED       [ 53%]
tests/test_diffusion_modules.py::TestConv1dBlock::test_output_matches[2-16-4-8-3-2] PASSED          [ 54%]
tests/test_diffusion_tf.py::test_initialization PASSED                                              [ 56%]
tests/test_diffusion_tf.py::test_forward_pass PASSED                                                [ 57%]
tests/test_diffusion_tf.py::test_loss_computation PASSED                                            [ 59%]
tests/test_diffusion_tf.py::test_p_losses_epsilon PASSED                                            [ 60%]
tests/test_diffusion_tf.py::test_p_losses_x0_prediction PASSED                                      [ 62%]
tests/test_diffusion_tf.py::test_sample_shape_ddim PASSED                                           [ 64%]
tests/test_diffusion_tf.py::test_action_clipping PASSED                                             [ 65%]
tests/test_diffusion_tf.py::test_denoised_clipping PASSED                                           [ 67%]
tests/test_mlp_diffusion.py::test_diffusion_mlp_shapes PASSED                                       [ 68%]
tests/test_mlp_diffusion.py::test_diffusion_mlp_extreme_times PASSED                                [ 70%]
tests/test_mlp_diffusion.py::test_diffusion_mlp_equivalence PASSED                                  [ 71%]
tests/test_model_save_load.py::test_save_and_load_model PASSED                                      [ 73%]
tests/test_model_save_load.py::test_train_save_load_compare PASSED                                  [ 75%]
tests/test_sampling.py::test_cosine_beta_schedule[10-0.008] PASSED                                  [ 76%]
tests/test_sampling.py::test_cosine_beta_schedule[100-0.1] PASSED                                   [ 78%]
tests/test_sampling.py::test_cosine_beta_schedule[1000-0.5] PASSED                                  [ 79%]
tests/test_sampling.py::test_cosine_beta_schedule[1-0.008] PASSED                                   [ 81%]
tests/test_sampling.py::test_extract[1-10] PASSED                                                   [ 82%]
tests/test_sampling.py::test_extract[4-100] PASSED                                                  [ 84%]
tests/test_sampling.py::test_extract[16-10] PASSED                                                  [ 85%]
tests/test_sampling.py::test_extract[1-1] PASSED                                                    [ 87%]
tests/test_sampling.py::test_make_timesteps[1-0_0] PASSED                                           [ 89%]
tests/test_sampling.py::test_make_timesteps[4-10] PASSED                                            [ 90%]
tests/test_sampling.py::test_make_timesteps[16-100] PASSED                                          [ 92%]
tests/test_sampling.py::test_make_timesteps[1-0_1] PASSED                                           [ 93%]
tests/test_sampling.py::test_numerical_stability PASSED                                             [ 95%]
tests/test_sampling.py::test_device_consistency PASSED                                              [ 96%]
tests/test_sampling.py::test_dtype_consistency PASSED                                               [ 98%]
tests/test_scheduler.py::test_schedulers PASSED                                                     [100%]

====================================================== 64 passed in 26.33s ======================================================
```