Extended Abstract

Motivation Recent advances in aligning Large Language Models (LLMs) with human preferences have highlighted Direct Preference Optimization (DPO) as a leading approach due to its simplicity and effectiveness. Unlike traditional reinforcement learning techniques, which require carefully engineered reward signals, DPO relies on pairwise preference labels, making it especially appealing in settings where explicit reward design is infeasible. While DPO has gained traction in language model fine-tuning, its applicability to robotics and control domains remains underexplored. This motivates our investigation into whether DPO can enhance performance in continuous control environments such as Gymnasium's `Humanoid` and `Pusher` tasks. These benchmarks present realistic robotic challenges, including high-dimensional action spaces and intricate system dynamics, offering a meaningful testbed for evaluating DPO's potential beyond language modeling.

Method We first trained base policies using behavior cloning on expert demonstration datasets for the `Humanoid` and `Pusher` environments from the Minari benchmark suite. Next, we collected both human-labeled and synthetically-generated preference pairs and fine-tuned the base models using Direct Preference Optimization. To benchmark our approach, we also trained PPO policies from scratch and compared their performance against our fine-tuned models. Lastly, we explored Active Learning to improve data efficiency in training our DPO models.

Implementation We first trained the base behavior cloning policies ($\pi_{BC}^{\texttt{Humanoid}}$ and $\pi_{BC}^{\texttt{Pusher}}$) on expert demonstrations from the Minari benchmark. The `Humanoid` model used a single-layer GRU with a linear mean head and global log-standard deviation, while `Pusher` used a two-layer MLP with 256 hidden units and ReLU activations. Both models were trained using the Adam optimizer.

We then used $\pi_{BC}^{\texttt{Humanoid}}$ and $\pi_{BC}^{\texttt{Pusher}}$ to roll out trajectories for both human and synthetic labeling. For the human-labeled dataset, a reviewer selected the preferred trajectory from each pair, while for the synthetic dataset, the trajectory with the higher environment reward was automatically chosen as preferred. We then used our labeled trajectories to train $\pi_{\textit{DPO Human}}^{\texttt{Humanoid}}$, $\pi_{\textit{DPO Synthetic}}^{\texttt{Humanoid}}$, $\pi_{\textit{DPO Human}}^{\texttt{Pusher}}$, and $\pi_{\textit{DPO Synthetic}}^{\texttt{Pusher}}$ with a single loop of DPO updates on the base policies.

We also trained PPO baseline policies ($\pi_{PPO}^{\texttt{Humanoid}}$ and $\pi_{PPO}^{\texttt{Pusher}}$) for both `Humanoid` and `Pusher` using the Stable-Baselines3 implementation with a two-layer MLP (64 units, $\tanh$ activations).

Results DPO improved upon the base policy by up to 16.6% in the `Humanoid` environment and up to 4.5% in the `Pusher` environment. Synthetic DPO performed around 2% better than human-labeled DPO in both environments. Active Learning experiments showed that the diversity of the trajectories in the preference pairs in low-data regimes have a huge impact on performance. For `Humanoid`, the importance sampling chose a very homogenous dataset and for `Pusher`, it chose a more diverse dataset, thus resulting in better performance over random.

Discussion Our experiments showed that DPO fine-tuning can significantly improve policy performance in both `Humanoid` and `Pusher`, with synthetic preferences slightly outperforming human-labeled ones—though the two agreed 95% of the time, reinforcing the validity of human feedback. Reference-free DPO introduced instability during training, suggesting the need for regularization, such as a KL divergence term, to improve generalization. Finally, while our active learning setup did not follow a traditional iterative loop, it still revealed promising directions for improving data efficiency in low-label regimes, especially in scenarios where preference uncertainty could guide sampling.

Conclusion Our results demonstrate that Direct Preference Optimization (DPO) is an effective method for fine-tuning low-level control policies in continuous control environments like `Humanoid` and `Pusher`. Both human-labeled and synthetic preference datasets led to performance improvements over behavior cloning and PPO baselines, with synthetic preferences performing slightly better. These findings highlight DPO's potential for real-world robotics applications, where aligning behavior with human preferences is critical and reward engineering is often infeasible.

Direct Preference Optimization for Low-Level Actions in Robotic and Simulation Learning

Kenneth Ma
Department of Computer Science
Stanford University
kenma25@stanford.edu

Parker Stewart
Department of Computer Science
Stanford University
parkers@stanford.edu

Thomas Yim
Department of Computer Science
Stanford University
yimt@stanford.edu

Abstract

Direct Preference Optimization (DPO) is a recent technique that fine-tunes policies directly from human preferences, bypassing the need for explicit reward functions or separate reward model training. While DPO has shown strong performance in language modeling, its effectiveness in low-level control tasks remains underexplored. In this project, we evaluate DPO in two continuous control environments—`Humanoid` and `Pusher`—by fine-tuning behavior cloning (BC) policies using both synthetic (reward-based) and human-labeled preference datasets. Our results show that DPO improves performance over the BC baseline in both tasks, with synthetic DPO slightly outperforming human-labeled DPO. Additionally, we investigate active learning strategies to improve data efficiency in preference collection. Overall, our experiments demonstrate that DPO can enhance policy quality in complex robotic environments and that preference-based tuning is a promising direction for reward-free learning.

1 Introduction

Direct Preference Optimization (DPO) has emerged as a state-of-the-art method for aligning Large Language Models (LLMs) with human preferences. Compared to other reinforcement learning methods, its simple implementation and effectiveness have made it common in modern fine-tuning workflows. Given the difficulty of defining accurate, dense reward functions encoding the attributes of a good or bad trajectory, DPO allows for alignment through a direct human-labeled preference on a pair of examples. However, demonstrations of its effectiveness in robotic or simulation-based applications are limited.

While defining a good reward function may be difficult, a labeler might still have an intuition for a better or worse series of actions - thus reducing the dependence on domain expertise to specify an objective. The goal of our project is to investigate whether DPO can be used to improve performance on the Gymnasium `Humanoid` and `Pusher` tasks by tuning the low-level primitive actions. These environments are representative of common challenges in robotics, including high-dimensional control and complex dynamics. The goals for these two tasks are notably different, as the `Humanoid` aims to sustain a running movement for as long as possible, whereas the `Pusher` seeks to complete its motion as efficiently as possible.

Stanford CS224R 2025 Final Report

2 Related Work

2.1 Proximal Policy Optimization (PPO)

Early policy gradient methods optimized agent behavior by increasing the likelihood of actions that led to higher rewards. While effective in theory, these methods often suffered from instability due to large policy updates during training. Proximal Policy Optimization (PPO) addresses this issue by introducing a clipped objective function that penalizes updates that deviate too far from the previous policy (Schulman et al. (2017)). This constraint stabilizes training while maintaining the sample efficiency of standard policy gradient methods. Due to its simplicity and robustness, PPO has become the foundation for many downstream reinforcement learning frameworks, particularly those that involve learning from human feedback. In both Reinforcement Learning from Human Preferences (RLHP) and Reinforcement Learning from Human Feedback (RLHF), PPO is commonly used to optimize the agent's policy with respect to a learned reward model trained from human feedback data. PPO's ability to perform stable updates from sampled experience makes it especially well-suited for such applications. However, PPO still relies on the existence of a good reward function to guide learning - which DPO addresses by replacing it with human preference.

2.2 Reinforcement Learning from Human Preferences (RLHP)

In many tasks, particularly those involving complex or subjective goals, defining a dense and accurate reward function is challenging or infeasible. Reinforcement Learning from Human Preferences (RLHP) addresses this problem by learning a reward model directly from pairwise human comparisons between short trajectory segments (Christiano et al. (2023)). Rather than predefining a numerical reward function, human evaluators are shown two behaviors and asked to select which one better aligns with the intended task objective. A neural network is then trained to predict these preferences, effectively learning a reward function that models human judgments. This learned reward is used to train the agent's policy through standard reinforcement learning algorithms, most commonly PPO. RLHP decouples reward specification from policy optimization, enabling agents to improve behavior in environments where hand-crafted reward signals are unavailable or misleading. This approach laid the foundation for modern alignment techniques in large-scale models, such as Reinforcement Learning from Human Feedback (RLHF), by showing that agents can be successfully guided through human preference data alone.

2.3 Direct Preference Optimization (DPO)

While RLHP and RLHF demonstrated that agents can be aligned with human preferences through reward modeling and reinforcement learning, these methods introduced additional complexity through multi-stage pipelines involving reward model training and policy optimization with PPO. Direct Preference Optimization (DPO) simplifies this process by eliminating the need for a learned reward function altogether (Rafailov et al. (2024)). Instead, DPO directly fine-tunes the policy using a contrastive objective that operates on pairwise human preferences. Given a prompt and two model outputs—one preferred and one rejected by a human annotator—DPO optimizes the policy to assign higher likelihood to the preferred response relative to the rejected one. This is achieved through a log-ratio loss that compares the likelihoods under the current policy and a frozen reference policy, encouraging improvement while anchoring the updated model to its original behavior. DPO retains the benefits of preference-based alignment while reducing computational overhead, increasing stability, and simplifying implementation. Empirically, DPO has been shown to match or exceed the performance of RLHF in aligning large language models, making it a compelling alternative for preference-guided fine-tuning workflows.

2.4 Hierarchical Preference Optimization (HPO)

While DPO enables direct fine-tuning of flat policies from human preferences, it does not address the challenges of long-horizon tasks where behavior must be decomposed into a sequence of subgoals. Hierarchical Preference Optimization (HPO) extends the DPO framework to hierarchical reinforcement learning by introducing a two-level policy architecture (Singh et al. (2024)). A high-level policy proposes subgoals, while a low-level policy learns to execute them. Crucially, HPO incorporates a regularization mechanism based on the low-level policy's value function, encouraging

the high-level policy to propose subgoals that are not only preferred by humans but also feasible for the system to achieve. This makes HPO one of the first frameworks to enable preference-driven learning of both high-level planning and low-level control in a unified, reward-free setting.

2.5 Active Learning for DPO

Fine-tuning with human preferences has demonstrated success in LLMs on downstream tasks, however these high-quality human preference datasets can be difficult to acquire. Being time-consuming and sometimes requiring expert labelers, assembing a human preference dataset for DPO can be prohibitively expensive. However, some training examples are more informative than others and will have a bigger impact on model performance. Active learning is a semi-supervised learning method that uses the current model or policy to estimate how informative an unlabeled example (or in the case of DPO a preference pair) would be to train the model. ActiveDPO is a method that uses the LLM itself to parameterize a reward model that then finds pairs with the largest absolute difference in reward to be labeled by a human (Lin et al. (2025)). So after investigating if DPO would improve performance on low-level robotic actions, we are curious to see if active learning could improve our data efficiency and achieve the same performance with fewer examples.

3 Method

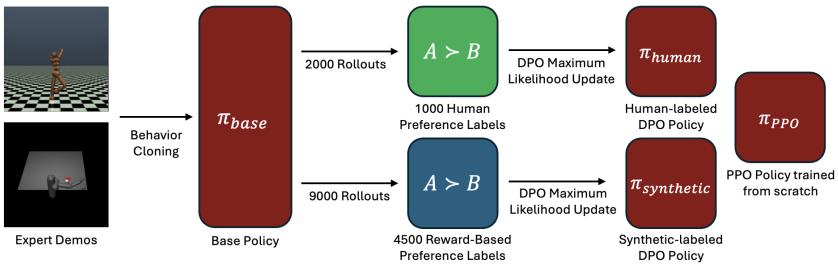


Figure 1: Method Overview.

3.1 Environments

3.1.1 Humanoid

This environment simulates the task of a humanoid figure trying to run forward and considers it a failure whenever it falls to the ground. The observation shape is (376,) and the action space is 17-dimensional with each value controlling the torque on a joint on the humanoid figure (restricted to [-0.4, 0.4]).

The reward defined in the environment consists of four components: `healthy_reward` + `forward_reward` − `ctrl_cost` − `contact_cost` where `healthy_reward` is the number of timesteps that the humanoid is alive, `forward_reward` measures the forward momentum of the figure, `ctrl_cost` is the magnitude of the action forces, and `contact_cost` is the magnitude of external contact.

3.1.2 Pusher

This environment simulates a 7-degree-of-freedom robotic arm that aims to push a cylinder into a target location. The observation shape is (23,) which consists of various joint angles, velocities, and positions of the arm, target, and cylinder. The action space is 7-dimensional with each value corresponding to the torque applied to each joint of the robotic arm (restricted to [-2.0, 2.0]).

The reward function is composed of three terms: `reward_near`+`reward_dist`−`reward_control` where `reward_near` is the distance from the fingertip of the robot arm to the cylinder, `reward_dist` is the distance from the cylinder to the target position, and `reward_control` is the magnitude of the torques applied.

3.2 Models

3.2.1 Humanoid

We used a Gated Recurrent Unit (GRU) RNN as the model for the `Humanoid` environment. This was chosen because of the high-dimensional and unstable dynamics of the `Humanoid` task, where there are more dependencies on past timesteps to help make decisions about the next action. It has a single-layer GRU with 256 hidden units and outputs a 17-dimensional vector that represents the mean for a Normal distribution that is used with a learnable log standard deviation shared across timesteps. This distribution is then used to sample the elements of the next action.

3.2.2 Pusher

For the pusher task, we used a simpler multilayer perceptron (MLP) model since there are less complex dynamics in the pusher environment and the current observation should be enough information to condition on for sampling the next action. This had two fully connected layers with 256 hidden units and ReLU activations. Similarly, this outputs a vector of means as well as a learnable log standard deviation parameter that we use for the Normal distributions we sample the actions from.

3.3 Behavior Cloning

Our base model is a behavior cloning (BC) policy using trajectories collected from experts in the `Humanoid` and `Pusher` environments in the Minari benchmark datasets.

Given a set of sequences containing observations and actions: $\left\{(o_{1:T}^{(i)}, a_{1:T}^{(i)})\right\}_{i=1}^{B}$, we train our policy π_θ with negative log likelihood loss.

$$\mathcal{L}_{\text{NLL}} = -\frac{1}{B}\sum_{i=1}^{B}\frac{1}{T}\sum_{t=1}^{T}\log \pi_\theta\left(a_t^{(i)} \mid o_{1:t}^{(i)}\right)$$

3.4 Direct Preference Optimization

Direct Preference Optimization can be implemented with or without an explicit reference policy. We implement a modified version of DPO called Reference-Free DPO. In the original equation. the function compares the log-likelihood ratio between our current policy π_θ and the fixed behavior-cloned base model on preferred and rejected trajectories.

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta\log\frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]$$

However, this penalizes model changes that deviate too far from the reference base model that we trained with behavior cloning. And after visualizing some rollouts from both the `Humanoid` and `Pusher` tasks after Behavior Cloning, we wanted a version of DPO that would allow for more exploration and thus generalization. This results in Reference-Free DPO (Meng et al. (2024)) which directly compares the log-likelihoods of the two trajectories.

$$\mathcal{L}_{\text{DPO}}(\pi_\theta) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\left(\log\pi_\theta(y_w \mid x) - \log\pi_\theta(y_l \mid x)\right)\right)\right]$$

Note that we no longer keep track of π_{ref} so we can simplify our training by no longer maintaining the frozen base model. It also makes training more computationally efficient as we do not need to re-evaluate the probabilities of the reference models.

3.5 Proximal Policy Optimization

We separately trained a Proximal Policy Optimization (PPO) policy in each environment to serve as a relative benchmark. PPO is an actor-critic reinforcement learning algorithm that improves training stability by preventing large policy updates (Schulman et al. (2017)). It optimizes a clipped surrogate objective to constrain policy changes and reduce the risk of performance collapse.

The clipped objective function is:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

is the probability ratio between the new and old policies, and $\hat{A}_t$ is an estimate of the advantage function. PPO also typically includes a value function loss and an entropy bonus to encourage exploration:

$$\mathcal{L}_{\text{PPO}} = \mathcal{L}^{\text{CLIP}} - c_1\mathcal{L}^{\text{VF}} + c_2 S[\pi_\theta]$$

where $\mathcal{L}^{\text{VF}}$ is the squared error between the predicted and actual returns, and $S[\pi_\theta]$ is the policy entropy.

4 Experimental Setup

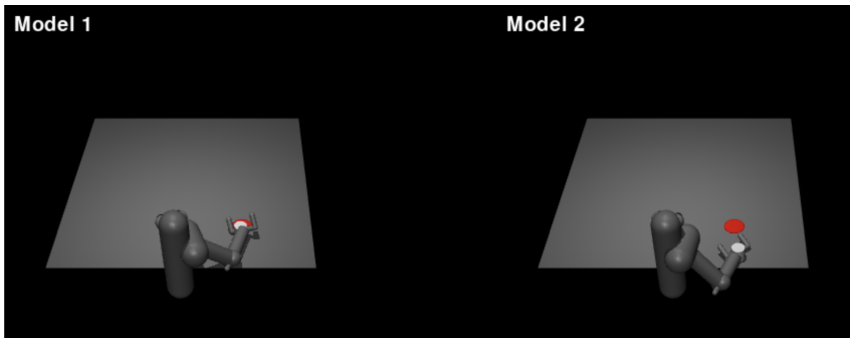


Figure 2: Pusher-v5 Trajectory Labeling

4.1 Behavior Cloning

To train the base BC models $\pi_{BC}^{\texttt{Humanoid}}$ and $\pi_{BC}^{\texttt{Pusher}}$, we first collected 1197 demonstrations from the `Humanoid` expert dataset with a maximum timestep of 1000 and 5000 trajectories from the `Pusher` expert dataset with a maximum timestep of 1000. We then generated sequences of length 32 to optimize behavior cloning learning.

For the `Humanoid` environment, we used the single-layer GRU described in section 3.2.1 to learn a linear mean head as well as a global log-standard deviation parameter. For the Pusher environment, we replaced the GRU with an MLP model with two fully connected 256-unit layers and ReLU activations, as described in section 3.2.2.

We trained the `Humanoid` model using a batch size of 64 and learning rate of $1\text{e}{-3}$ for 5 epochs, and the `Pusher` model using batch size of 64 and learning rate of $3\text{e}{-3}$ for 20 epochs. Both models used the Adam optimizer to improve training speed and learnability.

4.2 DPO Dataset Collection

We collected two types of datasets for both environments for DPO training: a synthetic dataset and a human-labeled dataset. In the synthetic dataset, the trajectory with the greater environment reward was automatically chosen as the preferred one. For the human-labeled dataset, we presented a reviewer with pairs of side-by-side trajectories (shown in Figure 2) and asked them to select the one they preferred. These preferences were recorded by pressing '1' to select the left trajectory or '2' to select the right. To prevent positional bias from influencing preference selections during human labeling, trajectories were randomly assigned to appear on the left or right side of the screen.

In both cases, the trajectory pairs were generated by rolling out π_{base} in the respective environments. We used a temperature parameter (0.33σ for `Humanoid` and 1.25σ for `Pusher`) to control the standard

deviation of the action distribution of the rollouts. This ensured that the dataset reflected meaningful variations in policy behavior, such that paired trajectories were similar but not directly identical.

For the human-labeled dataset, we reviewed and labeled 1,000 trajectory pairs. For the synthetic dataset, we rolled out 4,500 trajectory pairs. We stored the actions and observations for each labeled trajectory pair locally to be loaded for DPO fine-tuning.

4.3 DPO

To train the human-labeled DPO and synthetically labeled policies for the `Humanoid` and `Pusher` environments, we began by loading the 1,000 human-labeled and the 4,5000 synthetically labeled preference pairs respectively. We then loaded in the π_{BC} policies for `Humanoid` and `Pusher` environments.

We observed that validation performance began to decline after 10 epochs, likely due to overfitting on the training data. As a result, we limited training to 10 epochs. This corresponded to a total of 10,000 training timesteps on the human-labeled dataset for both `Humanoid` and `Pusher` and 45,000 timesteps on the synthetic dataset for each environment.

We then conducted a *coarse-to-fine hyperparameter search*: in the coarse stage, we randomly sampled a range of learning rates and β values to identify promising regions of the search space. In the fine stage, we performed a focused grid search around the best performing configurations to further refine the policy. This grid search ranged from 3.5×10^{-6} to 7.0×10^{-6} with a step of 0.5×10^{-6} for learning rate and 0.5 to 0.15 for β with a step of 0.05.

4.4 PPO

We separately trained PPO policies $\pi_{PPO}^{\texttt{Humanoid}}$ and $\pi_{PPO}^{\texttt{Pusher}}$ from scratch on the `Humanoid-v5` and `Pusher-v5` Gymnasium environments. To do so, we used the Stable-Baselines3 implementation of PPO (Raffin et al. (2021)), which uses an MLP policy with two 64-unit hidden layers and $\tanh$ activation functions for both the actor and critic networks.

For both environments, we trained for 10,000,000 total timesteps using 4 parallel actors in a vectorized environment, with a learning rate of $3\text{e}{-4}$, rollout length of 2048, batch size of 64, and 10 epochs. While training with a larger model for more timesteps would have produced better results, we opted for a smaller model to ensure that the train times were comparable between the all trained policies.

4.5 Active Learning

We use active learning to choose which unlabeled preference pairs should be labeled by a human then added to a dataset $\mathcal{D}$. We then evaluate how the model performs when fine-tuned with different sizes of $\mathcal{D}$. In the random sampling method, we randomly choose the next examples to append to the dataset. In importance sampling, we take the examples where the absolute difference in reward is greatest between the pairs. We test this in the low-data regime and outline the approach here:

Algorithm 1 ACTIVE LEARNING

Create a dataset $\mathcal{D}$ containing 5 random labeled preference pairs from $\mathcal{U}$
for *dataset_size* $\in \{10, 25, 50, 100\}$ **do**
  Use an active learning method to select (dataset_size $- |\mathcal{D}|$) new preference pairs from $\mathcal{U}$
  Append them to $\mathcal{D}$
  Train a new policy $\pi_{\text{active\_learning}}$
  Evaluate $\pi_{\text{active\_learning}}$

At each stage that we train, we use the same hyperparameters that worked best in the previous experiments where we used all the data. We also are keeping track of the best performance across all epochs of training.

5 Results

5.1 Quantitative Evaluation

Table 1: Reward Results

Method	`Humanoid` Mean Reward	`Pusher` Mean Reward
Base Model	7643.31	-27.14
Synthetic DPO	8915.89 (+16.6%)	-25.92 (+4.5%)
Human-Labeled DPO	8763.83 (+14.7%)	-26.36 (+2.9%)
PPO	293.48 (-96.2%)	-26.40 (+2.7%)

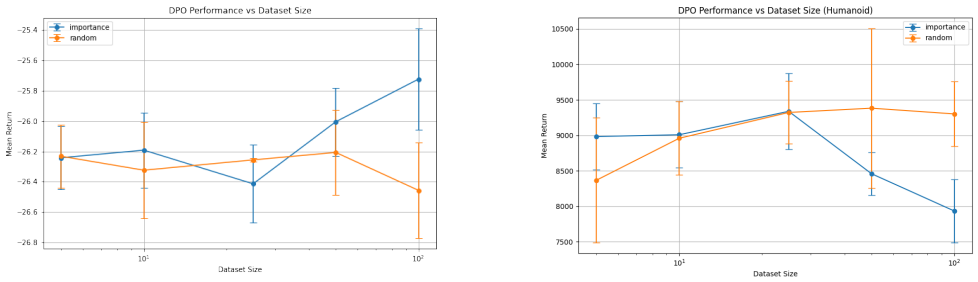
Our results demonstrate that DPO fine-tuning produces significant improvement in mean reward over the base models in both environments, while PPO trained using a similar-sized model for a similar number of timesteps performs far worse for the `Humanoid` environment. Synthetic labeling performs around 2% better than human labeling in both cases.

5.2 Qualitative Analysis

After visualizing and investigating sample trajectories after fine-tuning, we noticed `Humanoid` either starts from a position where it falls down immediately (reward of 200) or it picks up momentum and is able to run until we reach the maximum number of timesteps in the environment (reward of 10,000). Thus the `Humanoid` mean reward can be treated as an upper bound for the success rate (i.e. the baseline model was succeeding 76% of the time). Once it gets into a state of moving, the form is pretty much identical to the expert policy and it can keep moving, so DPO fine-tuning appears to be helping it move from a wider variety of starting positions.

`Pusher` does not have an equivalent failure case like falling down in the `Humanoid` task so the improvement was more subtle. During labeling, we would choose the trajectories with smoother and faster movement to the center of the goal zone, and that is the improvement we noticed when visualizing some trajectories after fine-tuning.

5.3 Active Learning



(a) Pusher (b) Humanoid

Figure 3: Average reward versus dataset size using Active learning on the `Humanoid` and `Pusher` tasks for dataset sizes of [5, 10, 25, 50, 100].

Although we hypothesized that importance sampling would get a higher mean reward than random sampling in a lower example regime for both environments, it is interesting to note that importance sampling outperformed random sampling in `Pusher` and was worse in `Humanoid`.

The performance difference at 5 examples can be attributed to different initializations, but there is significant performance deviations once we hit 100 examples. This is likely due to how similar trajectories were in the humanoid example. $\mathcal{D}$ for `Humanoid` only consisted of preference pairs where

one ran until the max timesteps and the other fell immediately. Thus we have a very homogenous fine-tuning dataset which is harming performance. However in `Pusher`, the failure cases were more diverse and the action space is a lot simpler so importance sampling performed better.

6 Discussion

We see that DPO fine-tuning can produce significant improvements to an already highly successful policy in both the `Humanoid` and `Pusher` environments. Due to the well-defined reward function in each environment and a larger synthetically-labeled dataset, synthetic labeling performed slightly better than human labeling. It is important to note, however, that synthetic and human labeling agreed 95% of the time, indicating that human labeling remains an effective way to select preferences, even when a well-defined reward function is not present.

With our formulation of Reference-Free DPO, training was notably unstable, with stagnant or decreasing returns across epochs despite decreasing loss. This suggests that our loss function would benefit from a KL divergence term, which would provide regularization against overfitting and improved generalization during training. Future work will investigate the implementation of this term and whether it will improve training stability.

The PPO models trained from scratch did not perform as well as DPO fine-tuning in both environments. This is largely due to the fact that we used smaller model sizes and less environment steps than optimal in order to have comparable train times between experiments. It is likely that, with this lightweight model, PPO settled into a local optimum for the `Humanoid` environment, leading to a stagnation in reward.

Furthermore, our active learning approach was dissimilar to a standard active learning experiment. Since our reward was defined by the environment, that does not change as we incrementally grow the dataset. A more realistic active learning experiment would take the mean uncertainty of the policy in choosing those actions in both preference pairs so it can see examples it might be less confident in. This then incorporates the model changes with the growing dataset. However, our experiment still gives us interesting insights on how we might approach training models in low-data regimes.

7 Conclusion

We found that Direct Preference Optimization (DPO) can be used to tune low-level trajectories in complex continuous control environments such as `Humanoid` and `Pusher`, leading to measurable improvements over behavior cloning baselines. Both human-labeled and synthetically generated preference datasets were effective in guiding policy improvement, with synthetic preferences yielding slightly better performance in our experiments.

Our results suggest that DPO is a viable alternative to traditional reinforcement learning approaches like PPO, particularly in scenarios where reward functions are sparse, hard to define, or misaligned with desired behavior. Additionally, even in scenarios where reward functions are well defined, behavior-cloning with DPO fine-tuning tends to outperform PPO. Furthermore, our exploration of a reference-free variant of DPO enabled greater policy flexibility, although at the cost of increased training instability. Finally, preliminary experiments with active learning showed promise in reducing the number of human-labeled examples required for effective fine-tuning, indicating a valuable direction for future work.

These findings have important implications for real-world robotics, where designing accurate reward functions is often infeasible. By enabling learning directly from human preferences, DPO offers a framework for training agents to perform complex tasks in alignment with human intent. This approach reduces reliance on hand-crafted reward engineering, making it easier to deploy adaptive, fine-tuned robotic systems in real-world environments where user preferences of behavior are critical.

8 Team Contributions

- **Kenneth Ma:** Trained the base models for the `Humanoid` and `Pusher` tasks using Behavior Cloning. Setup and trained both tasks with PPO. Labeled preference pairs for DPO training on both `Humanoid` and `Pusher`, and fine-tuned `Humanoid` DPO model.

- **Parker Stewart:** Created tool allowing us to label preference pairs for DPO training and labeled many examples. Worked on pipeline for DPO training and fine-tuned `Pusher` DPO model.
- **Thomas Yim:** Set up pipeline for DPO training. Conducted active learning and dataset size experiments. Labeled preference pairs for DPO training on both `Humanoid` and `Pusher`. Generated the reward-based preferences pairs

Changes from Proposal We have stayed close to the original assignments in the proposal.

References

Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2023. Deep reinforcement learning from human preferences. arXiv:1706.03741 [stat.ML] `https://arxiv.org/abs/1706.03741`

Xiaoqiang Lin, Arun Verma, Zhongxiang Dai, Daniela Rus, See-Kiong Ng, and Bryan Kian Hsiang Low. 2025. ActiveDPO: Active Direct Preference Optimization for Sample-Efficient Alignment. *arXiv preprint arXiv:2505.19241* (May 2025). `https://arxiv.org/abs/2505.19241` Accepted to ACL 2025.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. SimPO: Simple Preference Optimization with a Reference-Free Reward. arXiv:2405.14734 [cs.CL] `https://arxiv.org/abs/2405.14734`

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290 [cs.LG] `https://arxiv.org/abs/2305.18290`

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. `http://jmlr.org/papers/v22/20-1364.html`

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] `https://arxiv.org/abs/1707.06347`

Utsav Singh, Souradip Chakraborty, Wesley A. Suttle, Brian M. Sadler, Anit Kumar Sahu, Mubarak Shah, Vinay P. Namboodiri, and Amrit Singh Bedi. 2024. Hierarchical Preference Optimization: Learning to achieve goals via feasible subgoals prediction. arXiv:2411.00361 [cs.LG] `https://arxiv.org/abs/2411.00361`