A. Network Architectures

A.1. Autoencoders

The policy and dynamics autoencoders are parameterised by Transformers using stacked self-attention and point-wise, fully connected layers for the encoder, and a fully connected feed-forward network for the decoder.

Encoders: The encoders consist of one layer composed of two sublayers, followed by another fully connected layer. The first sublayer, is a single-head self-attention mechanism, and the second is a simple fully connected feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization and dropout. We use a dropout of 0.1 for all experiments. To facilitate these residual connections, all sublayers in the model, as well as the embedding layers, produce outputs of dimension $d_{model} = 64$. The second layer of the encoder projects the output of the first layer into the embedding space (from d_{model} to d_{emb}).

The **policy encoder** takes as input a set of state-action pairs (s_t, a_t) from an full trajectory and outputs an embedding for the policy.

The **dynamics encoder** takes as input a set of state-action-next-state tuples (s_t, a_t, s_{t+1}) from a full trajectory and outputs an embedding for the dynamics.

The dimension of both the policy and dynamics embedding is $d_{emb} = 8$ for all environments, with the exception of swimmer which uses a dynamics embedding of dimension 2.

Decoders: The decoder is a simple fully connected feed-forward network with three layers and ReLU activations after the first two layers.

The **policy decoder** takes as input the state of the environment and the policy embedding (outputted by the policy encoder) and outputs an action (i.e. the predicted action taken by the agent).

The **environment decoder** takes as input the state of the environment, an action, the dynamics embedding (outputted by the dynamics encoder) and outputs a state (i.e. the predicted next state in the environment).

The dimensions of the states and actions depend on the given environment.

A.2. The Policy-Dynamics Value Function

The Policy-Dynamics Value Function (PD-VF) takes as inputs the initial state of the environment, as well as a policy embedding and a dynamics embedding, and outputs a scalar representing the predicted expected return.

PD-VF is parameterised by a fully connected feed-forward network. First, the environment state and dynamics embedding are concatanated and passed through a linear layer with output dimension 64 followed by a ReLU nonlinearity. The second layer also has output dimension 64 but is followed by a hyperbolic tangent nonlinearity. The output of the second layer is then passed through another linear layer with output dimension equal to the square of the policy embedding dimension which is 64 in this case. Then, the output of this is rearranged in the form of a lower triangular matrix L. This matrix is used to construct a Hermitian positive-definite matrix A using the Cholesky decomposition, $A = LL^T$. Finally, the value outputted by the network is obtained by computing $z_\pi^T A z_\pi$, where z_{pi} is the policy embedding.

A.3. Baselines

All the pretrained PPO policies as well as all the baselines (except for CondPolicy as explained below) and the ablations use the same actor-critic network architecture. Both the actor and the critic are parameterised two-layer fully connected networks with hidden size 64 and hyperbolic tangent nonlinearities after each layer. Note that the weights are not shared by the two networks. The critic layer has another linear layer on top that outputs the estimated value. The actor network also has a linear layer on top that outputs a vector with the same number of dimensions as the action space. The actions are sampled from a Gaussian distribution with diagonal covariance matrix and means defined by the vector outputted by the actor network. The CondPolicy baseline has a similar architecture. The only difference is the first layer of both the actor and the critic, which has a larger input dimension due to the fact that these networks also take as input the policy embedding (along with the environment state).

B. Training Details

For experiments on Spaceship and Swimmer, we use only $N_d = 1$ steps to infer the dynamics embedding, while for Ant-wind we use $N_d = 2$ and for Ant-legs we use $N_d = 4$. Note that in all four domains, we only need a few steps to infer the environment dynamics, which allows us to quickly find a good policy for acting during the rest of the episode. Consequently, this results in good performance when evaluated on a single episode.

First, we have the **reinforcement learning phase**, in which we pretrain 5 different initializations of PPO policies in each of the 20 environments in our distribution (both those used for training and those used for evaluation). We train all policies for $3e6$ environment interactions, which we have found to be enough for all of them to converge to a stable expected return.

Then, in the **self-supervised learning phase**, we use the policies pretrained on the training environments (75 policies for each domain) to generate trajectories through the training environments. For each policy-environment pair, we generate 200 trajectories, half of which are used for training the policy and dynamics autoencodesr and the rest are used for evaluation. We train the autoencoders on this data for a maximum of 200 epochs and we save the models with the lowest evaluation loss. Note that the autoencoders are never trained on trajectories generated in the evaluation environments or by policies pretrained on those environments, but only on data produced by interactions with the training environments.

Once we have the pretrained policy and dynamics autoencoders, we use them for learning the policy-dynamics value function in the **supervised training phase**. To do this, we again generate 40 trajectories in the training environments (using only the policies pretrained on those environments). Half of these trajectories are used for training the PD-VF, while the rest are used for evaluation. In our experiments, we have found 20 trajectories from each policy-environment pair to be enough for training the model. For each trajectory, a policy embedding is obtained by passing the full trajectory through the policy encoder. Similarly, a corresponding dynamics embedding is obtained for each trajectory by passing the first few N_d transitions of that trajectory through the dynamics encoder. The initial state and the return of that trajectory are also recorded. Now we have all the data needed for training the PD-VF with supervision. The PD-VF takes as inputs the initial state, the policy and dynamics embeddings and outputs a prediction for the expected return (corresponding to acting with that policy in the given environment). It is trained with ℓ_2 loss using the observed return. For the initial training stage of the PD-VF, we use 200 epochs, while for the second stage that include data aggregation for the value function and policy decoder, we use 100 epochs. The second stage is repeated a maximum of 20 times (each training for 100 epochs). We select the model that obtains the lowest loss on the evaluation data (out of all the models after each stage). We use this model for probing performace on the evaluation environments.

C. Hyperparameters

For training the PPO policies, as well as the baselines and ablations, we searched for the learning rate in [0.0001, 0.0003, 0.0005, 0.001] and found 0.0003 to work best across the boad. The entropy coefficient was set to 0.0, value loss coefficient 0.5, number of PPO epochs 10, number of PPO steps 2048, number of mini batches 32, gamma 0.99, and generealized advantage estimator coefficient 0.95. We also linearly decay the learning rate. These values were not searched over since they have been previously optimized for MuJoCo domains and have been shown to be robust across these environments.

For MAML, we used the best hyperparameters found in the original paper for MuJoCo, so meta batch size of 20, 10 batches, and 8 workers.

For the dynamics autoencoder, we did a grid search over the learning rate in [0.0001, 0.001, 0.01] and found 0.001 to be best for the dynamics and 0.01 to be best for the policy. We also searched for the right batch size in [8, 32, 256, 2048] and found 8 to work best for the dynamics and 2048 for the policy. We also did grid searches over $d_{emb} \in [2, 8, 32]$ and found $d_{emb} = 8$ for the policy autoencoders and $d_{emb} = 2$ for the dynamics autoencoders (except for Ant, in which we use $d_{emb} = 8$. We also searched for the hidden dimension of the transformers $d_{model} \in [32, 64, 128]$ and found $d_{model} = 64$ to work best for both the policy and the dynamics embeddings.

For the value function, we tried different values for the number of epochs for the initial training phase $N_{ep,1} \in$ [1000, 500, 200, 100] and for the second training phase $N_{ep,2} \in$ [500, 200, 100] and we found 200 and 100 (i.e. each of the 20 data aggregation stages has 100 epochs) to work best, respectively. We also tried different learning rates from [0.0005, 0.001, 0.005, 0.01] and found 0.005 to be the best. Similarly, we tried batch sizes in [64, 128, 256] and found 128 to be the best.

All the results shown in this paper are obtained using the best hyperparameters found in our grid searches.

D. Environments

D.1. Spaceship Environment

The source code contains the Spaceship domain that we designed, which is wrapped in a *gym* environment, so it can be easily integrated with any RL algorithm and used to evaluate agents.

The task consists of moving a spaceship with a unit point charge from one end of a 2D room through a door at the other end. The action space consists of a fixed-magnitude force vector that is applied at each timestep. The room contains two fixed electric charges that deflect/attract the ship as it moves through the environment.

At the beginning of each episode, the agentś location is initialized at the center-bottom of the room with coordinates (2.5, 0.2). The target door is always located at the center-top of the room with coordinates (2.5, 5.0). The size of the room is 5, the size of the door is 1, and the temporal resolution is 0.3 (i.e. the time interval used to compute the next position of the spaceship given the current location and the applied force). The observation consists of the spaceshipś 2D location in the room (whose coordinates can be any real number between 0 and 5, the size of the room) and the action consists of the 2D force applied by the agent. The episode ends either when the agent hits a wall, exits the room through the door, or the agent has taken more than 50 steps in the environment. At the end of an episode, the agent receives reward that decreases exponentially with its distance to the target door. The decay factor is set to 3.0. For all other steps, the agent receives no reward. The distribution of dynamics is a centered circle with radius 1.5.

D.2. MuJoCo Environments

For Swimmer we use a circle with radius 0.1 to sample the environment dynamics, while Ant-wind uses a radius of 4.0. For all three domains with continuous distribution of dynamics (i.e. Spaceship, Swimmer, and Ant-wind), we sample 15 environments for training and we hold out 5 for evaluation. The evaluation environments have dynamics covering a closed interval from the distribution thus testing the ability of the model to extrapolate (rather than intrapolate) to different dynamics. The Ant-wind domain has a total of 16 environmets, 4 of which are used for evaluation.

E. Evaluation

In this section, we describe in detail the evaluation method and how the results reported here are obtained. For each trained model (i.e. PD-VF, a baseline or an ablation) and for each (unseen) test environment, we use that model to obtain a full trajectory through the given environment. This is repeated 10 times and the average return of the 10 runs is recorded. Then, we compute the mean and standard deviation (of this average return) across 5 different seeds for each model. These are the statistics shown in Figures 4 and 5.

To generate the t-SNE plots, we generated 10 trajectories for each policy-environment pair, including both the training and the evaluation ones. The encoders are used to obtain policy and dynamics embeddings corresponding to each trajectory. Then, t-Distributed Stochastic Neighbor Embedding (t-SNE) with perplexity 30 is applied to produce Figures 6 and 7. Figure 6 shows the t-SNE for the dynamics embedding, where each point is colored by the environment in which the corresponding trajectory (used to obtain that dynamics embedding) was collected. Conversely, Figure 7 shows the t-SNE for the policy embedding, where each point is colored by the policy which generated the corresponding trajectory (used to obtain that policy embedding).

F. Analysis of Learned Embeddings

Figure 11 shows a t-SNE plot of the learned policy embeddings for Spaceship, Swimmer, and Ant (from left to right). The top and bottom rows color the embeddings by the policy and environment that generated the corresponding trajectory, respectively. Trajectories produced by the same policy have similar embeddings, while those generated in the same environment are not necessarily close in this embedding space. This shows that the policy embedding preserves information about the policy while disregarding elements of the environment (that generated the corresponding embedded trajectory).

Similarly, Figure 10 shows a t-SNE plot of the learned dynamics embeddings on the three continuous control domains used for evaluating our method. The top row colors each point by the corresponding environment used to generate the trajectory (from which the embedding is inferred), while the bottom row colors each point by the corresponding policy. One can see that the embedding space retrieves the true dynamics distribution and preserves the smoothness of the 1D manifold.

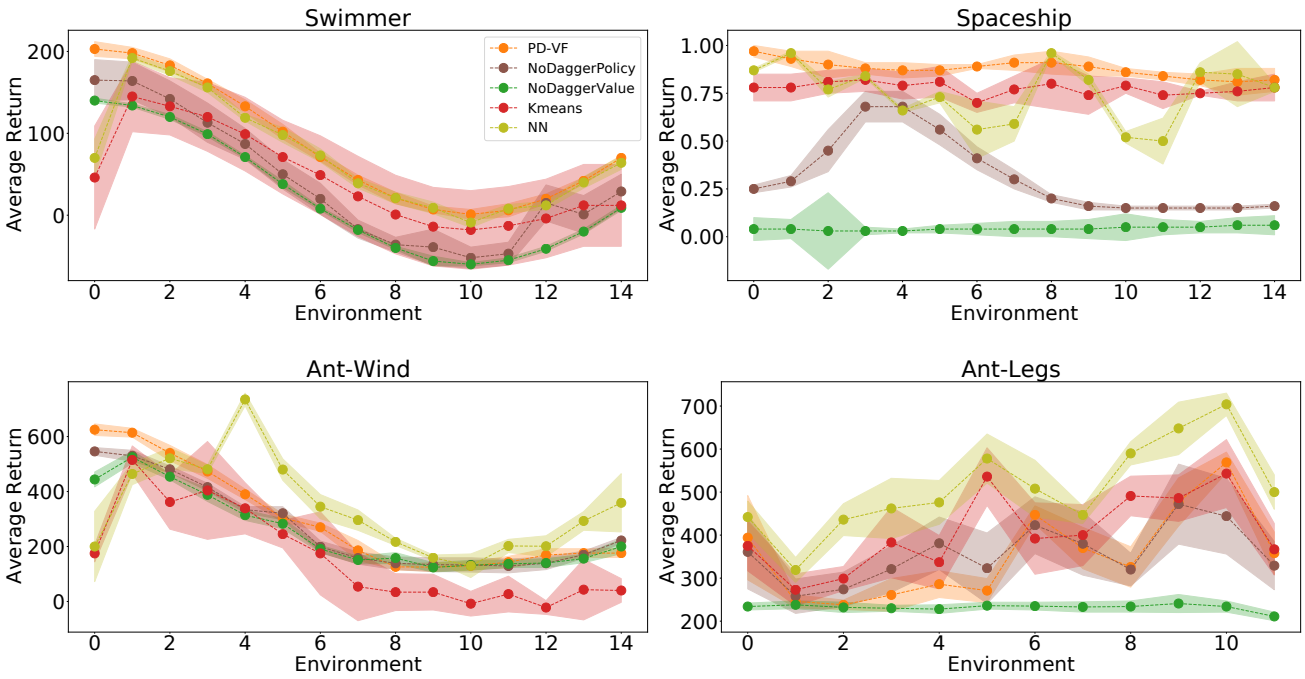

Figure 9. **Train Performance.** Average return on train environments in Swimmer (top-left), Spaceship (top-right), Ant-wind (bottom-left), and Ant-legs (bottom-right) obtained by PD-VF and a few ablations, namely NoDaggerPolicy, NoDaggerValue, Kmeans, and NN. PD-VF is comparable with or outperforms the ablations on the train environments. While some of these ablations perform reasonably well on the environments they are trained on, they generalize poorly to unseen dynamics.

Importantly, this analysis shows that the learned policy and dynamics embeddings are generally disentangled (i.e. information about the dynamics is not contained in the policy space and vice versa). This is important as we want the dynamics space to mostly capture information about the transition function and similarly, we want the policy space to capture variation in the agent behavior. The only exception is the dynamics space of Ant-wind, which contains information about both the environment and the policy. This is because in this environment, the policy is dominated by the force applied to the body of the ant, whose goal is to move forward (while incurring a penalty proportional to the applied force). Thus, depending on the wind direction in the training environment, the agent learns to apply a force of a certain magnitude, a characteristic captured in the embedding space. When evaluated on environments with different dynamics, that policy will still apply a similar force. Our experiments indicate that even if the dynamics space is not fully disentangled (yet it contains information about the environment), the PD-VF is still able to make effective use of the embeddings to find good policies for unseen environments and even outperform other state-of-the-art RL methods.

G. The Challenge of Transfer

In this section, we emphasize the fact that the family of environments we designed pose a significant challenge to current state-of-the-art RL methods. To do this, we train PPO policies on each of the environments in our set (until convergence) and evaluate them on all other environments. The results show that any of the policies trained in this way can drastically fail in other environments (with different dynamics) from our training and test sets. This demonstrates that our set of environments provides a wide range of dynamics and that a single policy trained in any of these environments does not generalize well to the other ones. Moreover, when evaluated on a single environments, the performance across the pretrained policies varies greatly, illustrating the diversity of collected behaviors (both optimal and suboptimal). This analysis further supports the need for learning about multiple policies (and their performance in various environments) in order to generalize across widely different scenarios (or dynamics in this case).

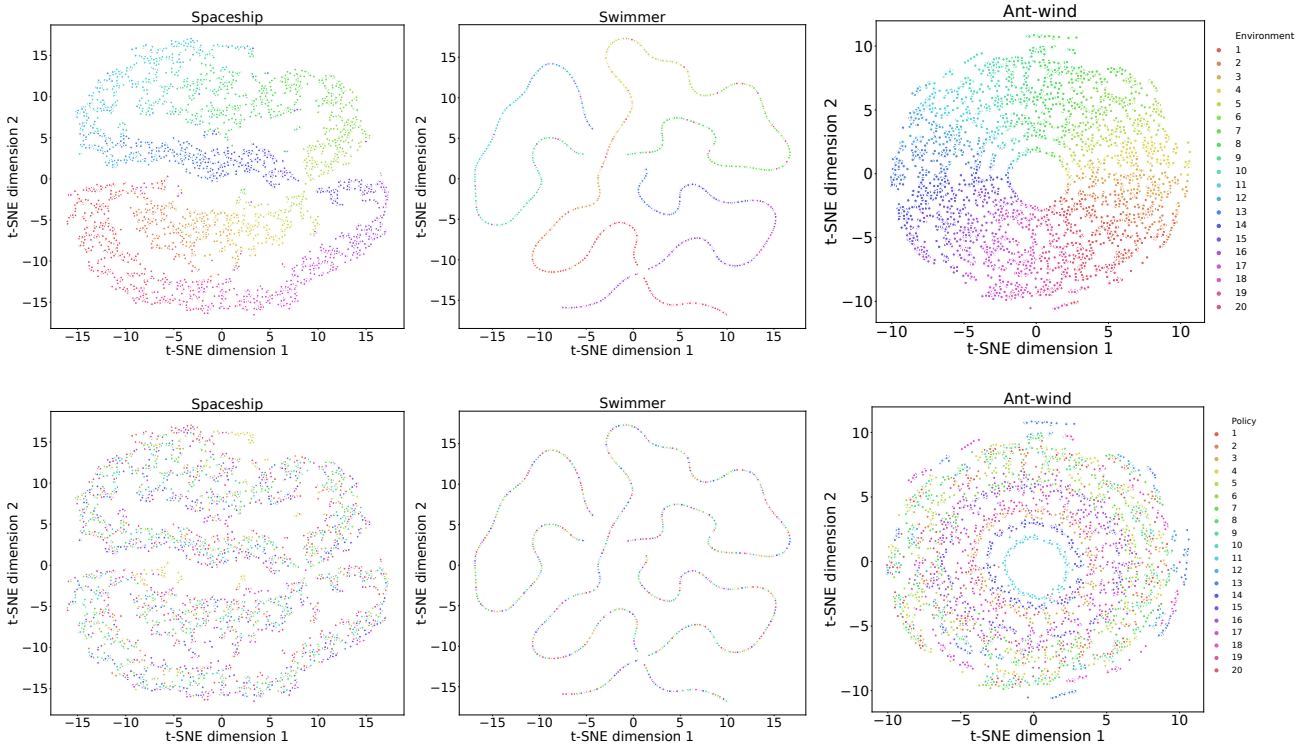

Figure 10. t-SNE plots of the learned environment embeddings z_d for Spaceship, Swimmer, and Ant-wind (from left to right). The points are colored by the *environment* (top) and *policy* (bottom) used to generate the trajectory of the corresponding dynamics embedding.

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
P1	598	354	128	372	291	95.9	-51.8	-50.9	-246	-299
P2	512	461	503	349	228	135	-1.44	-46.8	-28.1	-271
P3	689	620	593	500	334	80	0.4	-152	-51	-258
P4	654	665	557	519	29.9	180	177	-9.41	-90.5	-218.5
P5	935	962	947	930	853	648	429	287	155	-52.9
P6	811	838	794	778	710	600	386	247	123	-5.14
P7	624	659	408	451	351	474	394	82.9	151	55.9
P8	500	470	442	393	321	468	410	315	209	124
P9	303	326	297	295	265	254	243	238	11.1	223
P10	293	54.8	294	293	250	226	200	200	180	-1.28
P11	473	236	212	218	243	144	83.9	132	107	136
P12	266	264	268	242	214	181	55.6	72.7	239.4	240
P13	422	669	612	527	401	270	205	128	68.5	103
P14	436	362	424	366	259	296	97.3	55.7	24.9	-2.44
P15	420	484	264	125	131	66.7	44.5	13.1	5.43	35.1
P16	671	769	573	270	189	212	153	96.4	19.7	0.290
P17	784	793	683	600	56.9	200	56.8	12.4	4.4	19.8
P18	755	703	564	213	170	129	58.1	2.17	-103	-43.7
P19	182	593	415	65.5	250	112	25.8	37.1	-94.9	-19.2
P20	297	589	518	350	134	76.3	6.55	-51.5	-185.4	-11.2

Table 1. **Performance of PPO policies on the Ant-wind domain.** A row shows the mean episode return of a single policy on all environments, while a column shows the mean episode return of all policies on a single environment. This table contains performance on the first 10 environments.

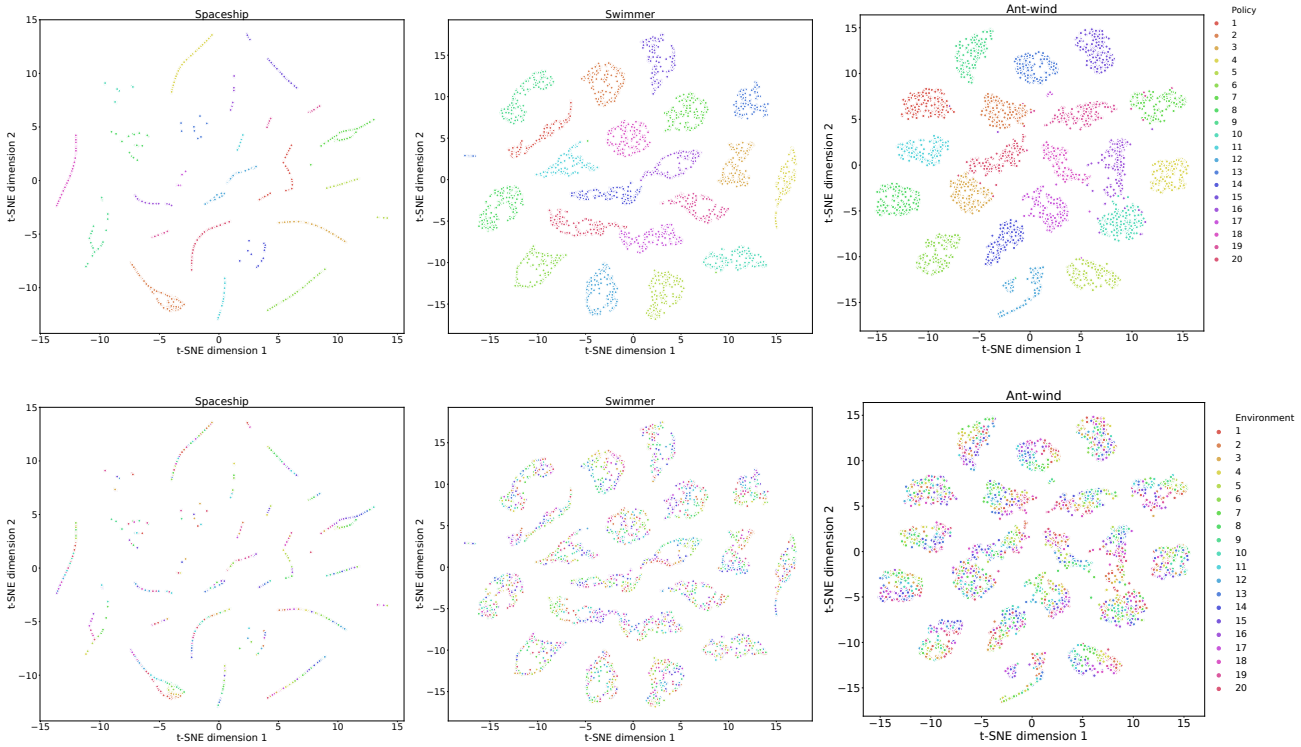

Figure 11. t-SNE plots of the learned policy embeddings z_π for Spaceship, Swimmer, and Ant-wind (from left to right). The points are colored by the *policy* (top) and *environment* (bottom) used to generate the trajectory of the corresponding policy embedding.

	E11	E12	E13	E14	E15	E16	E17	E18	E19	E20
P1	-336	-136	-55.2	-7.47	-8.11	168	262	473	498	603
P2	-42.2	-101	-28.3	-112	14.2	132	109	345	510	545
P3	-54.1	-10.1	-232	-6.25	59.9	120	207	136	563	372
P4	-279	-278	-135	-4.81	5.49	113	276	150	484	196
P5	-264	-236	-69.8	14.0	77.6	248	457	602	813	634
P6	20.9	-112	98.7	147	14.9	194	321	524	611	639
P7	0.700	-5.74	-61.6	9.51	89.0	267	212	412	579	536
P8	39.6	-33.5	11.7	53.9	21.5	206	218	29.9	469	509
P9	7.05	216	225	224	248	253	284	266	314	281
P10	176	-159	5.98	5.03	57.8	20.6	77.0	287	249	539
P11	92.8	148	206	244	277	351	280	485	556	582
P12	235	235	232	122	139	151	165	299	272	265
P13	84.6	171	230	285	312	300	328	523	640	504
P14	45.2	159	126	302	369	363	537	457	512	396
P15	-4.98	-70.8	41.3	360	571	654	687	546	393	537
P16	-7.77	19.7	63.4	248	457	600	740	493	731	804
P17	-20.0	-102	-8.87	69.8	254	444	515	658	749	703
P18	-93.7	-59.9	-11.6	125	275	434	577	679	781	781
P19	-10.1	-16.9	1.38	18.6	86.5	126	372	508	489	652
P20	-118	-62.4	-125	-81.7	20.1	82.9	131	195	341	392

Table 2. **Performance of PPO policies on the Ant-wind domain.** A row shows the mean episode return of a single policy on all environments, while a column shows the mean episode return of all policies on a single environment. This table contains performance on the last 10 environments.

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
P1	139.17	133.61	119.29	96.78	69.8	38.3	7.88	-19.59	-41.18	-54.88
P2	197.01	191.23	176.68	154.75	125.97	95.2	64.06	36.33	13.79	-1.1
P3	140.38	135.81	122.1	100.15	72.56	41.97	11.69	-15.47	-37.79	-51.59
P4	139.01	134.82	121.15	98.86	71.07	41.13	10.26	-17.24	-39.25	-54.68
P5	215.38	209.66	196.33	174.77	147.62	115.54	84.68	59.61	36.2	22.31
P6	207.08	201.55	186.96	165.49	138.86	107.64	77.12	49.94	28.6	14.55
P7	210.29	205.73	191.61	169.42	141.29	112.15	81.69	53.65	32.87	18.08
P8	213.98	209.7	198.61	179.11	152.56	124.49	95.18	67.78	45.47	32.82
P9	206.51	201.71	190.56	170.55	142.16	114.04	84.64	56.6	33.8	19.84
P10	204.85	201.04	186.64	168.29	141.0	113.45	80.69	53.39	33.46	19.9
P11	197.34	193.67	182.94	164.05	137.93	109.44	80.39	52.76	32.46	18.77
P12	204.92	201.11	186.85	166.49	138.03	108.22	79.32	50.95	29.97	16.23
P13	202.86	204.16	174.7	174.07	139.94	130.12	88.65	45.72	24.26	15.94
P14	205.89	201.64	187.07	166.63	138.54	108.87	77.48	51.66	29.76	15.68
P15	209.19	186.31	190.71	168.87	142.23	114.88	82.87	56.37	35.82	20.97
P16	214.29	204.26	188.0	165.51	140.3	109.66	80.35	56.25	37.05	23.41
P17	202.78	197.79	183.82	160.93	133.64	103.02	71.89	44.94	22.01	9.32
P18	202.66	204.15	190.16	167.73	139.81	109.03	77.82	51.18	29.4	14.83
P19	208.89	204.35	191.05	168.31	139.9	109.16	79.06	51.99	28.93	14.33
P20	141.37	136.28	122.22	100.69	73.13	42.51	12.4	-15.19	-37.6	-51.74

Table 3. **Performance of PPO policies on the Swimmer domain.** A row shows the mean episode return of a single policy on all environments, while a column shows the mean episode return of all policies on a single environment. This table contains performance on the first 10 environments.

	E11	E12	E13	E14	E15	E16	E17	E18	E19	E20
P1	-59.39	-54.24	-40.67	-19.18	9.2	39.2	69.81	97.3	119.73	134.49
P2	-6.11	-0.47	13.55	36.02	65.06	96.16	126.34	154.61	177.55	191.34
P3	-57.27	-52.51	-39.14	-16.39	10.56	40.38	71.77	98.78	121.09	135.31
P4	-58.16	-53.38	-39.35	-18.19	9.62	40.21	70.66	98.47	120.61	134.47
P5	16.13	20.32	32.89	56.59	85.07	114.87	144.05	173.91	196.76	210.27
P6	9.8	13.8	27.56	49.58	76.67	107.47	137.03	164.94	188.26	201.71
P7	14.63	19.66	34.54	56.04	82.39	114.16	143.69	171.16	190.65	205.65
P8	25.48	33.05	45.05	68.32	94.65	124.24	153.37	179.63	199.71	210.72
P9	15.39	19.33	31.88	54.41	79.43	110.51	139.44	165.05	186.49	200.29
P10	12.91	16.72	32.59	52.4	79.08	107.42	138.15	161.44	184.87	199.23
P11	12.84	17.28	30.41	50.77	76.78	104.17	132.33	156.83	178.78	191.66
P12	11.56	16.06	29.37	51.48	78.29	108.94	136.79	164.79	187.09	201.86
P13	3.07	-8.63	30.02	59.98	72.13	110.56	128.87	148.73	173.19	192.25
P14	10.65	16.23	30.17	52.24	79.07	109.76	139.88	166.59	187.43	202.78
P15	14.82	21.48	35.28	57.38	82.5	114.19	144.17	169.13	190.07	203.95
P16	18.92	23.92	36.31	57.05	88.57	116.78	147.6	172.97	196.18	209.08
P17	4.5	9.35	23.38	45.9	72.68	103.65	133.52	161.88	183.18	197.69
P18	10.91	14.36	29.04	49.94	78.39	109.27	140.76	167.58	189.9	202.89
P19	9.76	14.41	27.79	49.65	76.61	109.48	140.12	167.97	188.71	203.91
P20	-56.61	-52.49	-38.36	-16.25	11.95	42.14	72.88	100.43	122.74	136.87

Table 4. **Performance of PPO policies on the Swimmer domain.** A row shows the mean episode return of a single policy on all environments, while a column shows the mean episode return of all policies on a single environment. This table contains performance on the last 10 environments.

	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10
P1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.97	0.97	0.92	0.92	0.90	0.91	0.93	0.95	0.99	0.95
P3	0.97	0.92	0.85	0.85	0.86	0.86	0.86	0.91	0.94	0.94
P4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P5	0.61	0.61	0.60	0.58	0.57	0.55	0.53	0.51	0.50	0.49
P6	0.96	0.90	0.87	0.87	0.88	0.90	0.91	0.92	0.96	0.95
P7	0.83	0.89	0.93	0.96	0.97	0.96	0.94	0.89	0.84	0.80
P8	0.84	0.84	0.84	0.82	0.81	0.79	0.78	0.77	0.75	0.74
P9	0.97	0.93	0.90	0.87	0.86	0.86	0.87	0.87	0.87	0.86
P10	0.91	0.90	0.88	0.87	0.87	0.88	0.89	0.92	0.96	0.98
P11	0.77	0.78	0.79	0.80	0.82	0.78	0.72	0.76	0.79	0.80
P12	0.67	0.56	0.39	0.46	0.58	0.41	0.39	0.46	0.49	0.47
P13	0.38	0.36	0.32	0.26	0.30	0.25	0.32	0.32	0.33	0.40
P14	0.82	0.79	0.76	0.74	0.73	0.72	0.72	0.73	0.75	0.76
P15	0.67	0.66	0.65	0.63	0.62	0.62	0.61	0.61	0.61	0.61
P16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P17	0.86	0.82	0.80	0.78	0.77	0.78	0.79	0.82	0.85	0.87
P18	0.69	0.68	0.66	0.65	0.64	0.63	0.63	0.63	0.63	0.63
P19	0.80	0.81	0.81	0.73	0.75	0.78	0.81	0.79	0.78	0.90
P20	0.96	0.94	0.90	0.88	0.87	0.86	0.86	0.86	0.85	0.83

Table 5. **Performance of PPO policies on the Spaceship domain.** A row shows the mean episode return of a single policy on all environments, while a column shows the mean episode return of all policies on a single environment. This table contains performance on the first 10 environments.

	E11	E12	E13	E14	E15	E16	E17	E18	E19	E20
P1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P2	0.89	0.85	0.82	0.79	0.79	0.79	0.81	0.84	0.87	0.92
P3	0.93	0.91	0.86	0.83	0.82	0.82	0.84	0.86	0.91	0.95
P4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P5	0.49	0.49	0.49	0.50	0.51	0.53	0.55	0.57	0.59	0.61
P6	0.91	0.85	0.82	0.79	0.78	0.78	0.80	0.82	0.86	0.92
P7	0.74	0.70	0.67	0.65	0.64	0.65	0.66	0.70	0.75	0.79
P8	0.73	0.71	0.70	0.70	0.70	0.71	0.73	0.76	0.79	0.82
P9	0.84	0.82	0.80	0.79	0.80	0.81	0.84	0.87	0.93	0.97
P10	0.94	0.90	0.86	0.84	0.83	0.83	0.85	0.88	0.90	0.92
P11	0.81	0.93	0.90	0.87	0.84	0.82	0.80	0.78	0.77	0.77
P12	0.46	0.83	0.61	0.44	0.77	0.74	0.64	0.59	0.59	0.63
P13	0.43	0.73	0.64	0.81	0.67	0.69	0.80	0.63	0.51	0.41
P14	0.79	0.80	0.82	0.85	0.86	0.89	0.91	0.91	0.89	0.85
P15	0.61	0.62	0.62	0.63	0.64	0.65	0.66	0.67	0.68	0.68
P16	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P17	0.90	0.91	0.92	0.93	0.94	0.96	0.98	0.98	0.94	0.90
P18	0.63	0.63	0.64	0.64	0.65	0.67	0.68	0.69	0.70	0.70
P19	0.91	0.89	0.87	0.93	0.89	0.86	0.83	0.81	0.80	0.80
P20	0.80	0.78	0.77	0.76	0.76	0.77	0.80	0.85	0.90	0.93

Table 6. **Performance of PPO policies on the Spaceship domain.** A row shows the mean episode return of a single policy on all environments, while a column shows the mean episode return of all policies on a single environment. This table contains performance on the last 10 environments.