A Appendix

A.1 Proof for Theorem 2.1

Proof. Let $\pi(a_t|s_0, a_0, \ldots, s_t, z) = p_{\pi_\mathcal{D}}(a_t|s_0, a_0, \ldots, s_t, I(\tau) = z)$ and let $p_{\pi_\mathcal{D}}(s_t|s_0, a_0, \ldots, a_{t-1}) = p_{\pi_\mathcal{D}}(s_t|s_0, a_0, \ldots, a_{t-1}, I(\tau) = z)$. Then,

$$
\begin{aligned}
&\mathbb{E}_{\tau \sim p_{\pi_z}(\tau|s_0)}[D(I(\tau), z)] \\
&= \sum_{a_0, s_1, \ldots, s_T, a_T} p_{\pi_z}(\tau|s_0) D(I(\tau), z) \\
&= \sum_{a_0, s_1, \ldots, s_T, a_T} \prod_t \pi(a_t|s_0, a_0, \ldots, s_t, z) p(s_{t+1}|s_0, a_0, \ldots, a_t) D(I(\tau), z) \\
&= \sum_{a_0, s_1, \ldots, s_T, a_T} \prod_t p_{\pi_\mathcal{D}}(a_t|s_0, a_0, \ldots, s_t, I(\tau) = z) p_{\pi_\mathcal{D}}(s_t|s_0, a_0, \ldots, a_{t-1}, I(\tau) = z) D(I(\tau), z) \\
&= \sum_{a_0, s_1, \ldots, s_T, a_T} p_{\pi_\mathcal{D}}(\tau|s_0, I(\tau) = z) D(I(\tau), z) \\
&= \mathbb{E}_{\tau \sim p_{\pi_\mathcal{D}}(\tau|s_0, I(\tau)=z)}[D(I(\tau), z)] \\
&= 0
\end{aligned}
$$

□

The first equality follows from the definition of expectation. The second equality follows by the definition of a trajectory (the joint probability over the sequence of states and actions) and by the chain rule. The third equality follows due to our assumptions. Finally, the last two equalities use the definition of a trajectory and expectation.

To show the other direction, assume $\mathbb{E}_{\tau \sim p_{\pi_z}(\tau|s_0)}[D(I(\tau), z)] = 0$.

Then, $p_{\pi_z}(\tau|s_0) > 0$ implies $D(I(\tau), z) = 0$ since distance $D(\cdot, \cdot)$ is non-negative, which implies that when a trajectory has a non-zero probability, then $I(\tau) = z$.

Therefore, $p_{\pi_z}(\tau|s_0) = p_{\pi_z}(\tau|s_0, I(\tau) = z)$, which finally implies that $p_{\pi_\mathcal{D}}(s_t|s_0, a_0, \ldots, a_{t-1}) = p_{\pi_\mathcal{D}}(s_t|s_0, a_0, \ldots, a_{t-1}, I(\tau) = z)$.

A.1.1 Regarding History Conditioning

We thank the authors of Yang et al. [35], who pointed out an error in the theory in a previous draft of this paper via a counter-example in Appendix C. Our previous proof incorrectly assumed that independence from past states and actions, via our choice of Markov policies and via the Markov assumptions in MDPs, held when conditioning on $I(\tau)$. This is not in general correct, and we have updated our theory to reflect that theoretical guarantees under our framework require a memory-based policy and for the conditioning trajectory statistics to be independent of the future state given the entire history of the trajectory, not just the most recent state and action.

We note that in practice, Markov policies work well with ESPER, as does optimizing trajectory statistics for independence given only the most recent state.

A.2 ESPER Generalizes Return-Conditioning

While ESPER is designed to fix the problem with RvS in stochastic environments, it still performs well in deterministic environments. In a deterministic environment when the dynamics model can already predict the next state perfectly with just the current state and action, only the policy reconstruction loss will be active. Therefore, the optimal clustering will have each trajectory in its own cluster and labeled with their original return, and ESPER reduces to return-conditioned RvS. In Figure 7 we show empirically that the performance of ESPER matches return-conditioned Decision Transformer in the deterministic D4RL Mujoco tasks [13].

Dataset	Environment	Target	ESPER (ours)	DT	CQL
Medium-Expert	hopper	3600	89.95±13.91	79.64±34.45	**110.0**
Medium-Expert	walker	5000	**106.87±1.26**	**107.96±0.63**	98.7
Medium-Expert	half-cheetah	6000	**66.95±11.13**	42.89±0.35	**62.4**
Medium	hopper	3600	50.57±3.43	**59.46±4.74**	**58.0**
Medium	walker	5000	69.78±1.91	69.7±7.12	**79.2**
Medium	half-cheetah	6000	42.31±0.08	42.32±0.39	**44.4**
Medium-Replay	hopper	3600	**50.20±16.09**	**61.94±16.99**	48.6
Medium-Replay	walker	5000	**65.48±8.05**	**63.77±2.82**	26.7
Medium-Replay	half-cheetah	6000	35.85±1.97	36.88±0.36	**46.2**

Figure 7: Results on D4RL Mujoco Tasks

Task	Return-Conditioned RvS (MLP)	CQL	ESPER (MLP) (Ours)
Gambling	-0.05 (0.27)	**1.0 (0.0)**	**1.0 (0.0)**
Connect Four	0.24 (0.15)	0.61 (0.05)	**0.99 (0.01)**
2048	0.56 (0.03)	0.7 (0.09)	**0.81 (0.05)**

Figure 8: ESPER performs similarly on benchmark tasks when using a simple MLP rather than a transformer for the policy.

A.3 ESPER with MLPs

As reported in Emmons et al. [2], a transformer is not necessary to get strong performance with RvS on many environments. We also find that we get similar results when training using a simple MLP with three hidden layers, as shown in Figure 8.

A.4 Benchmark Task Details

A.4.1 An Illustrative Gambling Environment

To clearly illustrate the issue with prior approaches when conditioning on outcome variables that are correlated with environment stochasticity, we run our approach on a simple gambling environment. This environment, illustrated in Figure 1, has three actions: one which will result in the agent gaining one reward, and two gambling actions where the agent could receive either positive or negative reward.

A.4.2 Multi-Agent Game: Connect Four

Connect Four is a popular two-player board game where players alternate in placing tiles in the hope to be the first to get four in a row. In this task, we consider a single agent version of Connect Four where the opponent is fixed to be a stochastic agent. This is a realistic setting, since in the real world, the single greatest source of stochasticity will likely be other agents that the agent is interacting with. An ideal agent will need to take this into account to make optimal decisions.

Since Connect Four can be optimally solved with search techniques, we set the opposing agent to be optimal[2], with a small chance that it won't place a piece in the rightmost column when it is optimal to do so. This creates an MDP with two ways to win: first, the agent can play optimally to the end of the game to guarantee a win (the first player always can win); second, to win quickly, the agent can place four pieces on the rightmost column with the hope that the opponent will not block (which happens with a low probability).

The offline dataset for this task is generated by using an epsilon-optimal agent with 50% probability and an exploiter agent that only places pieces on the right with 50% probability.

[2]We use the solver from https://github.com/PascalPons/connect4.

A.4.3 Stochastic Planning in 2048

2048 [23] is a single player puzzle game where identical tiles are combined in order to build up tiles representing different powers of two. With each move, a new tile randomly appears on the board, and the game ends when no moves are available. A strong 2048 agent will consider the different possible places new tiles will appear in order to maximize the potential for combining tiles in the future.

Since the vanilla version of 2048 can require billions of steps to solve with reinforcement learning [36], we modified the game[3] by terminating the episode when a 128 tile is created. The agent gets one reward for successfully combining tiles in order to reach 128 and zero reward otherwise.

The offline dataset for this task is generated using a mixture of trajectories from an agent trained with PPO [24] using the implementation in Stable Baselines 3 [37] and a random policy.

A.5 Training Details

A.5.1 Hyperparameters

The most important hyperparameters to tune for our method are the tradeoff between reconstructing actions and removing dynamics information from the clusters, controlled by β_{act}, and the number of clusters, controlled by `rep_size` and `rep_groups`. The trajectory representation (i.e. cluster assignment) is formed by sampling from `rep_groups` categorical variables of dimension `rep_size` / `rep_groups`. These values were tuned per environment using a simple grid search. Specific hyperparameter values for each environment can be found at Table 1.

Decision Transformer	
`batch_size`	64
`learning_rate`	5e-4
`policy_architecture`	Transformer
`embed_dim`	128
`n_layer`	3
`n_head`	1
`activation_fn`	ReLU
`dropout`	0.1
`n_head`	1
`weight_decay`	1e-4
`warmup_steps`	10000
`discount` γ	1
`eval_samples`	100
ESPER	
`learning_rate`	1e-4
`batch_size`	100
`hidden_size`	512
`policy.hidden_layers`	3
`clustering_model.hidden_layers`	2
`clustering_model.lstm_hidden_size`	512
`clustering_model.lstm_layers`	1
`action_predictor.hidden_layers`	2
`return_predictor.hidden_layers`	2
`transition_predictor.hidden_layers`	2
`activation_fn`	ReLU
`optimizer`	AdamW [38]
`normalization`	batch norm [39]

Table 1: ESPER hyperparameters

[3] We used the implementation of 2048 found at https://github.com/activatedgeek/gym-2048.

Gambling	
rep_size	8
rep_groups	1
β_{act}	0.01
β_{adv}	1
cluster_epochs	5
label_epochs	1
policy_steps	50000
Connect Four	
rep_size	128
rep_groups	4
β_{act}	0.05
β_{adv}	1
cluster_epochs	5
label_epochs	5
policy_steps	50000
2048	
rep_size	128
rep_groups	4
β_{act}	0.02
β_{adv}	1
cluster_epochs	4
label_epochs	1
policy_steps	50000
Mujoco	
rep_size	256
rep_groups	4
β_{act}	0.03
β_{adv}	1
cluster_epochs	5
label_epochs	5
policy_steps	100000

Table 2: Environment hyperparameters

A.5.2 Computation

Our experiments were run on T4 GPUs and running our algorithm took only around 1 hour per seed. We used PyTorch [40] and experiments were tracked using Weights and Biases [41].

A.6 Baselines

We used the codebase provided by the authors of Decision Transformer [1] to implement our experiments. Other than hyperparameters specific to ESPER, Decision Transformer and ESPER use the same hyperparameters found in Table 1.For Conservative Q-Learning (CQL) [4], we used the default implementation from d3rlpy, an offline deep RL library [42].

A.7 ESPER Pseudocode

We provide pseudocode for ESPER in algorithm 1 and detailed pseudocode (roughly following the syntax used in PyTorch [40]) for the clustering step of ESPER in algorithm 2.

Algorithm 1: ESPER

Data: Dataset $\mathcal{D}$ consisting of trajectories of states, actions, and rewards
for *cluster iteration* $k = 1, 2, \ldots$ **do**
    $s, a \leftarrow$ sample batch of trajectories from $\mathcal{D}$;
    $\text{assignments} \leftarrow \text{ClusterAssignments}(s, a)$;
    $\hat{a} \leftarrow \text{ActionPredictor}(s, \text{assignments})$;
    $\hat{s}_{t+1} \leftarrow \text{TransitionPredictor}(s, a, \text{assignments})$;
    Update cluster assignments by training $\hat{a}$ to predict a and $\hat{s}_{t+1}$ to *not* predict s_{t+1} by minimizing Equation 2.2;
    Train the $\text{TransitionPredictor}$ to predict next states by minimizing Equation 2.3;
Fit a model $f_\psi(I(\tau))$ to predict the average trajectory return $\hat{R}$ in each cluster;
Create dataset $\mathcal{D}'$ of states, actions, and average returns;
for *policy iteration* $k = 1, 2, \ldots$ **do**
    $s, a, \hat{R} \leftarrow$ sample batch of states, actions, and average returns from $\mathcal{D}'$;
    Train the policy $\pi(a|s, \hat{R})$ by minimizing Equation 2.5;

Algorithm 2: ESPER - Adversarial Clustering

```
# s, a, seq_len: states, actions, sequence lengths (in case of early termination)
# Models:
# encoder_mlp - MLP
# temporal_encoder - LSTM
# rep_mlp - MLP
# act_mlp - MLP
# dynamics_mlp - MLP

def ClusterAssignments(s, a):
    bsz, t = s.shape[:2]
    x = torch.cat((s, a), dim=-1).view(bsz, t, -1)
    x = torch.flip(x, dims=[1])
    x = torch.encoder_mlp(x)
    x, hidden = temporal_encoder(x, init_hidden())
    x = torch.flip(x, dims=[1])
    x = rep_mlp(x)
    cluster_assignments = F.gumbel_softmax(x)
    return cluster_assignments

def ActionPredictor(s, cluster_assignments):
    # cluster_assignments.shape = [bsz, t, -1]
    # For a timestep t, we sample a cluster assignment for a timestep
    # from 0, ..., t randomly
    past_assignments = sample_past_assignments(cluster_assignments)
    x = torch.cat((s, past_assignments), dim=-1)
    pred_next_action = act_mlp(x)
    return pred_next_action

def TransitionPredictor(s, a, cluster_assignments):
    # cluster_assignments.shape = [bsz, t, -1]
    # For a timestep t, we sample a cluster assignment for a timestep
    # from 0, ..., t randomly
    past_assignments = sample_past_assignments(cluster_assignments)
    x = torch.cat((s, a, past_assignments), dim=-1)
    pred_next_s = dynamics_mlp(x)
    return pred_next_s

# training loop
for (s, a) in dataloader:
    # get cluster assignments for the trajectories
    cluster_assignments = ClusterAssignments(s, a)
    # predict actions based on clusters
    pred_next_action = ActionPredictor(s, cluster_assignments)
    # predict state transitions based on clusters
    pred_next_s = TransitionPredictor(s, a, cluster_assignments)
    # optimize the clusters for action prediction and to hurt next state prediction
    cluster_loss = act_loss(pred_next_action, a) - state_loss(pred_next_s, s)
    cluster_loss.zero_grad(); cluster_loss.backward(); cluster_optimizer.step();
    # optimize the transition predictor
    dyn_loss = state_loss(pred_next_s, s)
    dyn_loss.zero_grad(); dyn_loss.backward(); dyn_optimizer.step();
```